2016

An investigation into Off-Link IPv6 host enumeration search methods

Clinton Carpene

2016

An investigation into Off-Link IPv6 host enumeration search methods

Clinton Carpene

Edith Cowan University

Copyright Warning

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

An Investigation Into Off-Link IPv6 Host Enumeration Search Methods

by

Clinton Carpene

A thesis presented for the degree of

Doctor of Philosophy

School of Science
Edith Cowan University
Perth, Western Australia
11th February, 2016

Abstract

This research investigated search methods for enumerating networked devices on off-link 64 bit Internet Protocol version 6 (IPv6) subnetworks. IPv6 host enumeration is an emerging research area involving strategies to enable detection of networked devices on IPv6 networks. Host enumeration is an integral component in vulnerability assessments (VAs), and can be used to strengthen the security profile of a system. Recently, host enumeration has been applied to Internet-wide VAs in an effort to detect devices that are vulnerable to specific threats. These host enumeration exercises rely on the fact that the existing Internet Protocol version 4 (IPv4) can be exhaustively enumerated in less than an hour. The same is not true for IPv6, which would take over 584,940 years to enumerate a single network. As such, research is required to determine appropriate host enumeration search methods for IPv6, given that the protocol is seeing increase global usage.

For this study, a survey of Internet resources was conducted to gather information about the nature of IPv6 usage in real-world scenarios. The collected survey data revealed patterns in the usage of IPv6 that influenced search techniques. The research tested the efficacy of various searching algorithms against IPv6 datasets through the use of simulation.

Multiple algorithms were devised to test different approaches to host enumeration against 64 bit IPv6 subnetworks. Of these, a novel adaptive heuristic search algorithm, a genetic algorithm and a stripe search algorithm were chosen to conduct off-link IPv6 host enumeration. The suitability of a linear algorithm, a Monte Carlo algorithm and a pattern heuristics algorithm were also tested for their suitability in searching off link IPv6 networks. These algorithms were applied to two test IPv6 address datasets, one comprised of unique IPv6 data observed during the survey phase, and one comprised of unique IPv6 data generated using pseudorandom number generators. Searching against the two unique datasets was performed in order to determine appropriate strategies for off-link host enumeration under circumstances where networked devices were configured with addresses that represented real-word IPv6 addresses, and where device addresses were configured through some randomisation function.

Whilst the outcomes of this research support that an exhaustive enumeration of an IPv6 network is infeasible, it has been demonstrated that devices on IPv6 networks can be enumerated. In particular, it was identified that the linear search technique and the variants tested in this study (pattern search and stripe search), remained the most consistent means of enumerating an IPv6 network. Machine learning methods were also successfully applied to the problem. It was determined that the novel adaptive heuristic search algorithm was an appropriate candidate for search operations. The adaptive heuristic search algorithm successfully enumerated over 24% of the available devices on the dataset that was crafted from surveyed IPv6 address data. Moreover, it was confirmed that stochastic address generation can reduce the effectiveness of enumeration strategies, as all of the algorithms failed to enumerate more than 1% of hosts against a pseudorandomly generated dataset.

This research highlights a requirement for effective IPv6 host enumeration algorithms, and presents and validates appropriate methods. The methods presented in this thesis can help to influence the tools and utilities that are used to conduct host enumeration exercises.

I certify that this thesis does not, to the best of my knowledge and belief:

i) incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education;

ii) contain any material previously published or written by another person except where due reference is made in the text of this thesis; or

iii) contain any defamatory material.

Signed: . ..

Dated: 11/02/2016

Acknowledgments

Primarily I have to thank my fiancée Laura. You have been a driving force in my life, and have inspired me to aspire to do my best. Your continued love and support have made this possible. To my family, thank you for always being there for me with support and understanding when things have been rough. To list you all would require another thesis, so I'll just stick to the immediate family; Lee, Barbara, Brendon, Giuseppe, Sue, Karl, and Eve. You have all given me so much in life and I am eternally grateful.

To my principal supervisor Andrew, for helping me through what is a long, frustrating, and at times more of a mental and psychological journey than a physical one, thank you. Your blunt appraisal of my work has always been refreshing and simultaneously soul-crushing.

To my associate supervisors, Michael and Craig. Michael you have been a great source of wisdom about all things computing throughout my tenure. Craig, you have given me many opportunities since I began studying at ECU, and continue to put up with me for some reason. I cannot thank you enough.

To Eve and Laura (again), special mention must be made to your butchering of my drafts with highlighters and other marking instruments.

To ECU and in particular the School of Science, and the Security Research Institute, thanks for giving me the opportunity to undertake this degree and conduct this research.

Finally to Gary Hale, and Cisco systems for supporting this research through sponsorship, and also for providing equipment that has been used to conduct this research. Without that support and assistance this research would simply not have been possible.

Contents

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

Chapter 1

Introduction

It is increasingly commonplace for devices with varying purposes, such as computers, cars, mobile phones, even lightbulbs and fridges, to incorporate some access to the Internet. These devices depend on access to ever-decreasing network layer address resources. The current network communications protocol providing these addresses, Internet Protocol version 4 (IPv4), has almost exhausted its address supplies; with APNIC allocating its final 8 bit address block in 2014 (APNIC, 2015). Various systems, such as RFC-1918 private addresses (Rekhter, Moskowitz, Karrenberg, de Groot & Lear, 1996), and forms of Network Address Translation (NAT) have been introduced to assist in combatting the exhaustion problem. However, ultimately these measures serve to temporarily ameliorate the problem, rather than solve it. In 1994, a new protocol was approved to resolve the pending address exhaustion crisis; the Internet Protocol version 6 (IPv6).

IPv6 introduces a larger address space than its predecessor, IPv4. The increase is not insignificant, in fact the 128 bit IPv6 address space is 2^{96} times larger than that of IPv4. This increase in address space will provide the Internet with enough address resources to facilitate its expansion for the foreseeable future.

However, the increased address space introduces complications in locating nodes on a distant network using existing host enumeration techniques. In IPv6 there are enough potential unique addresses for 2^{64} nodes to exist on a subnetwork. Enumerating all hosts on a network by simply probing each possible address, as is commonplace with

IPv4, is no longer a feasible activity. With exhaustive searching currently infeasible, efforts must turn to more informed methods of enumeration.

1.1 Background to the problem

Host enumeration, also known as network discovery, host discovery or network scanning, is the act of identifying live nodes on a computer network. Host enumeration generally involves sending a probe, or series of probes to potential hosts on a network and measuring responses. Host enumeration has existed for a number of years as a means for agents to locate computers on networks. On a small network, such as a standard 24 bit IPv4 subnetwork, it may take an agent a matter of seconds to perform an exhaustive search of the address space and enumerate all of the hosts that are active on the network. In regards to an IPv6 network, where the recommended subnetwork size is 64 bits long (Hinden & Deering, 2006), a similar exercise would take many years, given current network bandwidth. To put this difference in perspective, at a rate of 1,000,000 probes per second, it would take less than a second to enumerate a 24 bit IPv4 network, whilst it would take approximately 584,942.42 years against a single 64 bit IPv6 network.

Enumerating host devices on computer networks can be scoped as either off-link or on-link enumeration. On-link discovery is when a device performs host enumeration against directly connected networks. In contrast, off-link refers to performing host enumeration against a network that is not directly connected. Specifically, this occurs when the agent performing host enumeration is on a separate link-layer broadcast domain to the host enumeration subject(s). Off-link host discovery is more commonplace across public networks, where it is often used as a tool for troubleshooting networks problems, locating and taking census of network resources, researching vulnerabilities, and attacking computer networks.

Host enumeration forms an important stage in vulnerability assessments (VAs) and network security assessments. VAs serve to identify potential security vulnerabilities in a system or group of systems by testing systems for their vulnerability to known threats. They are routinely conducted to assess the security profile of a network or

organisation. VAs can be performed proactively as a defence strengthening measure, or reactively to determine exposure to specific vulnerabilities. Common VA methodologies generally involve a phase dedicated to conducting host enumeration to identify devices in a network address space that are active (Braunton, 2005; McNab, 2007). A list of live nodes serves to restrict the scope of any further efforts in a vulnerability assessment to improve efficiency. Braunton (2005) provides a security assessment methodology that includes a phase dedicated to host enumeration. Braunton (2005) stresses the importance of conducting a thorough and well-documented host enumeration exercise as the foundation to a successful network security assessment.

Host enumeration techniques, and VAs also provide research opportunities. In 2014 a number of vulnerabilities were discovered in ubiquitous software packages, such as OpenSSL, bash and schannel. The Heartbleed, ShellShock and POODLE vulnerabilities were all publicly disclosed in 2014, and resulted in independent research efforts conducted to determine the exposure of each vulnerability. These efforts provided insight into the nature of the vulnerabilities, including longitudinal data of affected systems and remedial patching cycles.

Host enumeration techniques are often used in network support and troubleshooting. Network engineers and systems administrators commonly use network discovery tools and techniques to determine hosts that are active on their own computer networks. A computer network can experience frequent connections and disconnections of client devices. This is especially true in situations where bring your own device (BYOD) policies are unrestrictive or in the case of Internet of Things (IoT) systems. BYOD and IoT systems can host a large quantity of unmanaged devices. In such situations host enumeration can be used to take a census of the devices on computer network segments. Since host enumeration is a tool that is used to locate networked devices it is also often leveraged by malicious agents.

Malicious agents have used host discovery for many years (G. F. Lyon, 2009) to determine live nodes on networks when conducting penetration tests or unsolicited VAs. Once initial discovery is complete, an agent may target those machines and conduct an unauthorised vulnerability assessment in order to enumerate a system's vulnerabilities. They can then use other tools to exploit vulnerabilities to achieve their

goals in the attack. As an example, the aforementioned Heartbleed and ShellShock vulnerabilities could be enumerated by performing an exhaustive search of the IPv4 Internet (or a subset thereof) using specially crafted payloads, and then recording the response. These enumerated systems that have been confirmed to be vulnerable could then be further exploited to gain control of those systems.

Malware developers have also used host enumeration strategies in the past to create self-propagating worms such as the Conficker worm (Savagaonkar, Sahita, Nagabhushan, Rajagopal & Durham, 2005; Porras, Saidi & Yegneswaran, 2009). In such cases an infected computer will enumerate other potentially vulnerable hosts on a network and then attempt to transfer malware to them. These forms of worms were devastating to IPv4 computer networks before anti-virus software saw widespread adoption. Due to the IPv4 address space size, CPU processing power, and distributed enumeration worms could propagate very quickly.

Due to its large address space IPv6 has earned the reputation of being infeasible to enumerate using off-link host enumeration. An address space can be defined as a range of discrete addresses. In the context of the research problem, the address space is measured in bits. For IPv4 the address space of the entire protocol is 32 bits (approximately 4.3 billion addresses), while the IPv6 address space is 128 bits (approximately 340 undecillion addresses). The address space commonly used in subnetworks in IPv4 are substantially lower than that of IPv6. Common networks in IPv4 use 24 bits for the network portion and contain 256 possible addresses per subnet. IPv6 uses 64 bits for the network portion of the address which provides approximately 18 quintillion addresses per subnet. It is important to note the distinction between address types in IPv4 and IPv6. Both protocols have portions of the address space that are designated for public or global usage, and portions that are not for public usage. Henceforth, unless otherwise specified the IPv6 address range under consideration in this research is the global unicast address range (`2000::/3`). All publicly routable address ranges were considered when referring to IPv4.

On-link host enumeration is trivial to conduct in both IPv4 and IPv6 domains, since both protocols include provisions for hosts to discover their neighbours. Such facilities are required for communications between hosts to occur at all. In IPv4 the

Address Resolution Protocol (ARP), allows devices to query neighbours using link-layer broadcast frames. In IPv6, the Internet Control Message Protocol version 6 (ICMPv6), similarly to ARP, allows devices to discover their on-link neighbours. On-link discovery is usually achieved by querying common multicast addresses, for which each IPv6 enabled device is a member (such as the `all-nodes` address `ff02::1`). These methods do not apply to devices on different broadcast domains since such addresses are not routable outside their network. The ramifications of this protocol design means that devices attempting to perform host discovery from another network must employ differing tactics.

In contrast to on-link scanning, off-link scanning cannot simply be accomplished through queries to local broadcast or multicast address scopes. In such circumstances, agents seeking to discover hosts on neighbouring networks must employ network layer and above host discovery. Off-link discovery methods commonly used against IPv4 systems typically include an exhaustive linear search for small target sizes (e.g. for searching a single, 24 bit network). An exhaustive linear search is also possible (and commonplace) when searching the wider IPv4 Internet. Target randomisation techniques are often used instead to distribute the searching load evenly across the search space. Search distribution is employed to ensure the endpoints of the search attempt do not get overloaded with probes in the event that a single entity manages a contiguous address block. Utilities such as `nmap` (G. F. Lyon, 2009) and `masscan` (Graham, 2013c) employ randomisation functions to permute the entire address space randomly, rather than sequentially.

Current methods of off-link host enumeration in the IPv6 domain rely upon either performing a linear search of a subset of the address space, or targeting addresses that are commonly used in IPv6 deployments. A number of tools have been developed to accomplish the task of host enumeration against IPv6 networks using the limited algorithms discussed. These tools include the hacker's choice IPv6 suite's `alive6` program and the Chiron suite's IPv6 scanner. These applications both attempt to enumerate IPv6 networks through a hybrid approach of address pattern generation and linear searching.

1.2 Purpose of the Study

This research aims to fill the gap in knowledge that exists surrounding IPv6 host enumeration strategies. From the literature surrounding the topic it is evident that the problem of enumerating nodes on an off-link IPv6 network is considered infeasible. Currently there is minimal literature that provides methods for conducting off-link IPv6 host enumeration. Furthermore, no formal testing or validation of host enumeration methods that do exist have been performed. This study seeks to validate existing approaches to host enumeration, including those used against IPv4 and IPv6 networks. The study also seeks to devise and present new, efficient strategies that address the problem.

The literature in the domain does suggest that the protocol's implementation may assist with the formulation of searching methods that target areas of the address space efficiently. These methods may use heuristics to determine behaviour patterns in address allocation and exploit them to improve enumeration rates. Additionally, thus far machine learning methods have not been applied to the problem. Machine learning may provide another potential avenue for host enumeration search methods that have yet to be explored. Conventional search methods rely on generating static or deterministic target address lists. The adaptive nature of machine learning systems may improve the effectiveness of off-link host enumeration search methods for IPv6.

By exploring new methods to conduct off-link IPv6 host enumeration and testing the effectiveness of existing methods, this research aims to contribute validated methods of IPv6 host enumeration. These methods serve to aid in the efforts of future enumeration endeavours against IPv6 networks. It is an intended outcome of this research that the findings and discoveries will assist in improving the landscape for IPv6-related research efforts.

1.3 Research questions

The purpose of this research was to determine effective means to conduct host enumeration against IPv6 networks. The research questions were:

RQ1 *Can networking devices be enumerated on 64 bit IPv6 subnetworks using host discovery techniques?*

RQ2 *Are stochastic searching methods more efficient than deterministic searching methods when enumerating IPv6 hosts within a single 64 bit subnetwork?*

RQ3 *Do stochastic address allocation schemes within a single 64 bit subnetwork inhibit IPv6 host enumeration strategies?*

RQ4 *Can machine learning search methods be used to enumerate devices on a 64 bit IPv6 subnetwork?*

RQ5 *Are machine learning searching methods more efficient than non-machine learning based methods when enumerating IPv6 hosts within a single 64 bit subnetworks?*

1.4 Hypotheses

In order to answer the above research questions, hypotheses were formed. The primary hypotheses underpinning the research were:

H1 *"Search techniques are unable to enumerate networked devices on 64 bit IPv6 subnetworks."*

H2 *'Methods that employ random sampling do not perform better than methods that do not employ random sampling for IPv6 host enumeration."*

H3 *"Randomly generated interface identifiers do not affect the performance of IPv6 host enumeration search algorithms."*

H4 *"Search methods that employ machine learning are unable to enumerate networked devices on 64 bit IPv6 subnetworks."*

H5 *"Search methods that employ machine learning do not perform better than search methods that do not employ machine learning for IPv6 host enumeration."*

1.5 Research design

A quantitative study was designed and undertaken in an attempt to address the research hypotheses and provide answers to the research questions. Laboratory experimentation was chosen as an appropriate mode of inquiry for the research process.

The research was designed into five phases, involving:

1. Survey IPv6 usage: A survey into the real-world usage of IPv6 was conducted as a precursor to the study, and involved conducting a DNS enumeration against public IPv6 domain names. The survey data were used to influence the types of algorithms that were developed. The datasets used as the target networks in the research experiments were constructed in this phase of the research.

2. Generate search algorithms: The algorithms that would form the research's independent variables were designed in this phase. These algorithms drew influence from existing host enumeration algorithms, as well as algorithms that have not been previously applied to the problem.

3. Develop instruments and experiments, and perform pilot studies: The computer programs that would realise the testing and measuring of the research variables were constructed and tested in this phase.

4. Perform experiments: The research involved conducting experiments that applied algorithms to IPv6 address datasets. Each experiment consisted of two sub-experiments that exposed the subject algorithm to a different IPv6 dataset of valid addresses representing a configured IPv6 network. These sub-experiments were comprised of a number of simulations.

5. Data processing and analysis: The data collected from the experiments were collated and processed. Analyses were performed to test the research hypotheses using the observed results.

1.6 Assumptions

The author has made the assumption that the target hypothetical networks tested in this study are statically set for the duration of search attempts. This means that the networks do not have hosts connecting to or disconnecting from them. This distinction is important given that networks in real-world scenarios are often dynamic in nature, especially in networks that contain user devices (such as smartphones, laptop computers, workstations or tablets). Devices in these networks may connect and disconnect intermittently or sometimes permanently. This consideration was out scope for this research, and thus convergence rates of the algorithms are not considered within scope of this research. The datasets created to replicate IPv6 networks represent networks where all of the hosts are static. In real-world situations, the dynamic nature of computer networks would warrant investigation into optimisation of algorithm convergence rates. Such research could be undertaken in followup to this study.

It was also assumed that nodes on the hypothetical networks must respond to probing attempts if they exist. Effectively this means that if an address is included in the dataset, it is a live node. If an address is not included in the dataset, then it is not a live node, and no further consideration should be taken for it. Again, in a real-world situation there are a number of reasons why a node might not respond to a probe request. For example, the probe may be lost during transmission, it might get blocked by a firewall rule, or there could be unforeseen routing issues. However, these situations were not considered to be within the scope of the research.

This research was concerned only with Global Unicast IPv6 addresses, since they represent the publicly contactable address type offered by the IPv6 protocol under Hinden and Deering (2006), and are therefore a logical candidate for off-link host enumeration. Although other address types exist, some of which are publicly accessible, the global unicast address range are the most general type and most commonly used within the public Internet. Other specialty address types (such as the well known prefix, etc.) have nuances that would require specific handling (such as an increased network mask and static network bits), and have been excluded from the scope of this research. Additionally, the target IPv6 network(s) that experiments were conducted

against were 64 bit subnetworks (i.e. networks with a 64 bit network mask). 64 bits is the recommended subnetwork size by Hinden and Deering (2006) for global unicast networks, and was accepted as the standard size for this research.

For the purposes of the research it was assumed that the target network is bound by a one gigabit per second network connection, and that the source of the search attempt was bound by the same connection speed. Where appropriate for comparisons, a constant transmission rate of 1,000,000 generic probes per second has been declared. The rate of 1 million packets per second has been at least observed in Durumeric, Wustrow and Halderman (2013b) where the authors achieved a rate of 1.4 million packets per second, and in Graham (2013c) where 29 million packets per second were observed. The type of probe delivery method, or the probe payload, were not considered, since the search algorithms were the focus of the study.

1.7 Thesis structure

The remainder of the thesis is set out as follows:

Chapter Two, the Literature Review, reviews the research-oriented literature on host enumeration within IP networks. It contextualises this literature in terms of the problem of host enumeration across large address spaces, such as those that are greater than 32 bits. Outcomes from the review shaped the research questions by identifying gaps in knowledge.

Chapter Three, the Research Methodology and Design discusses the research methods and approaches used to perform the research. The main modes of inquiry are described, along with the epistemological backing for the research. The experimental process is discussed as well as the algorithms used to perform the host enumeration.

Chapter Four, the Results, presents the results gathered from the experiments that were conducted during the study and the outcomes of hypothesis testing. Of particular interest was the results of the linear search algorithms and the adaptive-heuristic search algorithms, which displayed strong results in their search operations against networks configured with real-world addresses.

Chapter Five, the Analysis and Discussion, explores the results gathered during the

research. In particular, the relationships between the results, their impact on testing the research hypotheses, and ultimately how they serve to answer the research questions is explored in this chapter.

Chapter Six, the Conclusions, summarises the contributions to knowledge made by the work explored in this thesis. Finally, this chapter suggests potential avenues for future research.

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK

Chapter 2

Literature Review

2.1 TCP/IP

2.1.1 Overview

The TCP/IP suite (also known as the Internet protocol suite) was developed in 1974 (Cerf, Dalal & Sunshine, 1974) when the need for globally interconnected networks was first realised. The first iterations of the Internet Protocol included the internetwork transmission control program which was described in RFC-675 by Cerf et al. (1974). The fourth version of the TCP/IP family was ratified in 1981 in RFC-791 (Postel, 1981).

TCP/IP gained traction after being named the protocol of the ARPANet by the US Department of Defence (DoD), a 500 node computer network, and the predecessor to the modern Internet in 1972, and securing military adoption in the 1980s. TCP/IP eventually became the de-facto standard networking protocol (Kessler, 2014). IPv4 is ubiquitous and is supported on nearly every network-aware device worldwide.

The TCP/IP protocol suite is based upon the TCP/IP model (formally known as DoD model (Kessler, 2014)) that uses four layers to describe communications between nodes. These four layers are;

- The Link layer, where protocols such as Ethernet and Token Ring provide a physical connection between devices on a network;
- The Internet layer, where protocols such as IPv4, IPv6 and IPSec allow devices

to communicate with each other beyond their physically connected network;

- The Transport layer, protocols such as TCP and UDP operate at this layer to provide end-to-end connectivity between nodes, and;
- The Application layer, where protocols transmit arbitrary data between each other. Protocols such as hypertext transfer protocol (HTTP) and simple message transmission protocol (SMTP) exist at the application layer (Kessler, 2014).

These four layers interoperate to complete the TCP/IP model, and provide the basis for computer networking.

The TCP/IP model is not the only framework for describing network communications. The four-layered TCP/IP model can be mapped to the Open Systems Interconnection's (OSI) seven layered networking model, as displayed in Figure 2-1. In the OSI model there are seven layers of abstraction defined for networking communications;

- The Physical layer, where electrical signals are transferred over a medium;
- The Data link layer, which, when paired with the Physical layer, is analogous to the Link layer in the TCP/IP model;
- The Network layer, which is synonymous to the Internet layer in the TCP/IP model;
- The Transport layer, which is synonymous to its TCP/IP counterpart;
- Finally, the Session, Presentation and Application layers, which are all concerned with processing arbitrary data streams, are combined to form the Application layer's counterpart in the TCP/IP model.

As previously stated, IPv4 and IPv6 operate at the Internet layer (layer two) of the TCP/IP model. These protocols provide a means for abstract communications to occur between devices over discontiguous physical networks ('Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model', 1994). IP networks are separated by boundaries known as subnetworks, which occur when an IP network is divided into smaller IP networks through the use of a subnet mask.

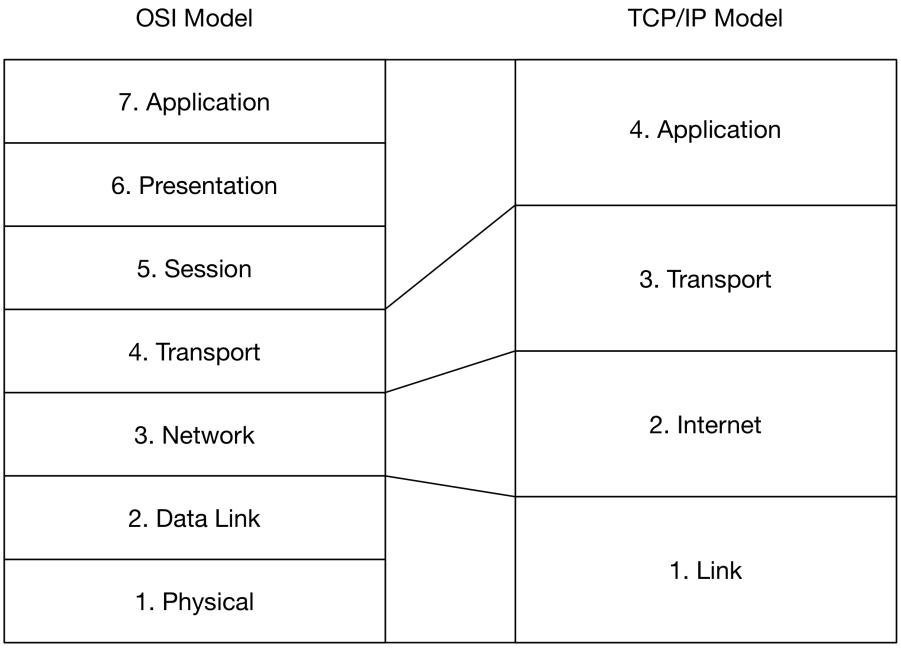


Figure 2-1: The seven layers of the OSI model (left) are comparable with the four layers of the TCP/IP model (right). The OSI Physical and Data link layers (layers 1 and 2, respectively), converge to form the layer 1 Link layer in the TCP/IP model, whilst the OSI Network layer (layer 3) is synonymous with TCP/IP Internet layer (layer 2). The transport layer is a direct translation from OSI layer 4 to TCP/IP layer 3. Finally OSI layers 5 to 7 (the Session, Presentation and Application layers) combine to form the TCP/IP model's Application layer (layer 4).

Subnet masks are binary streams that are applied to an IP address or network address in order to determine a start and end point of an IP network. Subnet masks are typically set to the binary value of one for all of the higher order bits from bit 0 until bit $n-1$ (where n is the number of network bits). The lower order bits (i.e. from bit n to bit 31) are set to the binary value of zero. As an example, in IPv4 a typical network size is 24 bits long. In this instance, the network's subnet mask would be represented as `255.255.255.0` (or `11111111.11111111.11111111.00000000` in dot-separated binary). When applied to an IP address, using a bitwise *and* operation, the result provides the network address (e.g. applying a bitwise *and* operation against an IP address `192.168.1.34` and the subnet mask `255.255.255.0` results in the network address of `192.168.1.0`) (Kessler, 2014). The binary masking system allows devices to determine the boundaries of networks in IP systems.

2.1.2 Address types

Devices participating in IP communications must have at least one unique IP address. IPv4 addresses are 32 bit values while IPv6 addresses are 128 bit values. In common presentation these values can be represented textually in many ways. It is especially true for IPv6, whose larger address space has necessitated compression techniques in order to reduce the size of represented addresses. IPv4 addresses can be represented as their unsigned integer value (a discrete number between 0 and 2^{32}), or by the representation standard noted in RFC-1123 (Braden, 1989) which dictates that an IPv4 address should be reduced to a string containing four chunks (octets), each which represents an eight bit unsigned integer, separated by periods.

IPv6 is less straightforward. The integer representation is similar to that of IPv4, with the only difference being the number of bits that comprise the address. Whilst the common representation for IPv6 addresses has undergone some reassessment since it was devised in RFC-2460 by Deering and Hinden (1998), the fundamental representation has not changed. IPv6 addresses should be represented as strings separated into eight chunks (hextets or quartets), each comprising a 16 bit hexadecimal string, separated by colons (e.g `2001:db8b:ccdd:eeff:1122:3344:5566:7777`). According to Kawamura and Kawashima (2010), leading zeroes must be dropped from each quartet, and a collapsing scheme can be used to compress multiple quartets of consecutive zeroes. As an example `1000:0:0:0:0:0:0:0001` would become `1::1`. In order to prevent ambiguities the collapsed notation can only appear once within an address. In cases where there are discontiguous quartets of zeroes, collapsing will occur at the largest consecutive number of zero value quartets. In the case of a deadlock, the highest order set of consecutive zeroes will be collapsed. For example `1000:0:0:1:0:0:0:1` would become `1000:0:0:1::1`, and `1000:0:0:1:1:0:0:1` would become `1000::1:1:0:0:1` (Hinden, Deering & Nordmark, 2003; Hinden & Deering, 2006; Kawamura & Kawashima, 2010).

It has been established that IPv6 addresses are discrete numbers between 0 and 2^{128}, and that when participating in IP networking, a node must have at least one unique address. It is not entirely true that there are 2^{128} possible public IPv6 addresses. In practice the address space has been partitioned to allow for future expansion of the

protocol, and reassessment of the addressing schemes. Under the current specifications detailed in Hinden and Deering (2006), IANA (2013) and IANA (2014b) the global unicast address space is `2000::/3`, which equates to 2^{125} possible addresses. The current major address space allocations for the IPv6 protocol are included in Table 2.1. This research is concerned only with addresses in the global unicast address space, since that is the address range that is publicly accessible, and most appropriate for off-link host discovery. From this point onwards, unless otherwise specified, references to IPv6 addresses or networks will relate to subjects of the global unicast IPv6 address space.

When using IPv6 there are a number of ways that assigning unique addresses to hosts can be achieved. A node can be configured with an IPv6 address either statefully or statelessly. A device can be provided with an address through means that require third party intervention (stateful addressing) or by autoconfiguring an address without requiring third party intervention (stateless addressing). It is important to reiterate that there are three components to an IPv6 address as defined in RFC-2460 by Deering and Hinden (1998). These components include the distinction between the global routing prefix of the address. The global routing prefix represents the higher order bits of an address, and can be further deconstructed to reveal the hierarchy of the address. Internet registries exist that allocate Internet resources to clients, including other Internet registers, and Internet service providers (ISP). There is also the distinction made for the subnet id (m) which denotes the subnetwork that an IPv6 address is a member of. The subnet id is a subset of the network portion of an IPv6 address. If following RFC-4291 (Hinden & Deering, 2006), the subnet id can be calculated by $64 - n$. Finally, there is the Interface Identifier which is made up of the lower order bits of the address (see Figure 2-2) calculated by $128 - n - m$.

The common stateful addressing schemes for IPv6 include the dynamic host configuration protocol version 6 (DHCPv6) and static addressing. DHCPv6 is similar to DHCPv4 to the extent that a server provides clients with unique addresses and other network resources defined in DHCP option fields (such as nameservers, time servers, etc.). Under DHCPv6 schemes, address allocation can happen using pools or through stochastic generation. For completeness, it is worth mentioning that there is also the Stateless DHCPv6 scheme. However, under this scheme DHCP is only used to deliver

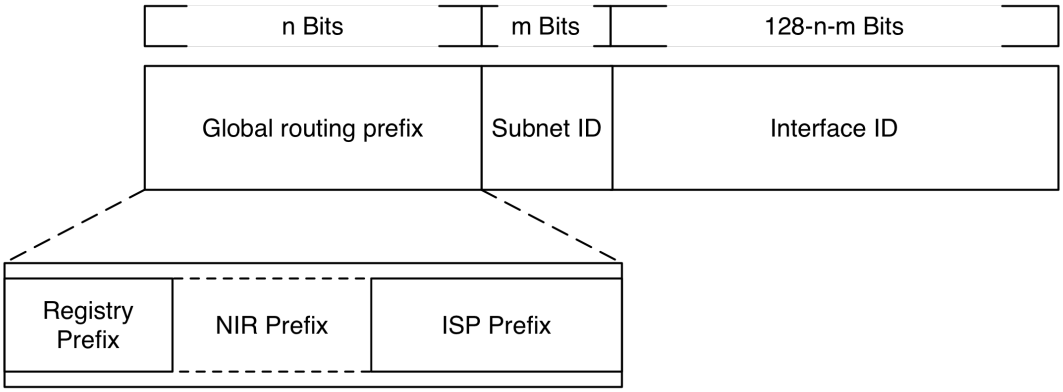


Figure 2-2: Global Unicast IPv6 address representation modified from RFC-3587 (Hinden, Deering & Nordmark, 2003) displaying the breakdown of the distinct portions of an IPv6 address. The global routing prefix (n bits) are the highest order bits of the IPv6 address. This can further be deconstructed to reveal the Internet service provider (ISP) prefix, and any Internet registry prefixes. The subnet ID (m bits) determines the subnetwork that an address is a part of. Finally, the Interface ID, the lowest order bits in the address, can be calculated by $128 - n - m$.

Table 2.1: The currently allocated IPv6 address ranges as specified by IANA (2013)

IPv6 Prefix	Prefix Allocation	Notes
0000::/8	Reserved by IETF	• ::1/128 is the IPv6 Loopback Address (equivalent to IPv4's 127.0.0.1) • ::/128 reserved for Unspecified Address • ::ffff:0:0/96 reserved for IPv4-mapped Address • 0000::/96 deprecated by RFC-4291 (Hinden & Deering, 2006). Formerly defined as the 'IPv4-compatible IPv6 address' prefix. • The 'Well Known Prefix' (64:ff9b::/96) is used to map IPv4 addresses to IPv6 addresses
0100::/8	Reserved by IETF	• 0100::/64 reserved for Discard-Only Address Block
0200::/7	Reserved by IETF	• Formerly OSI NSAP-mapped prefix, deprecated as of December 2004.
2000::/3	Global Unicast	• Publicly routable IPv6 addresses
fc00::/7	Unique Local Unicast	• Private IPv6 addresses, not publicly routable in the global Internet
fe80::/10	Link-Scoped Unicast	• Addresses used by nodes to communicate with devices on-link. Not routable outside origin link
fec0::/10	Reserved by IETF	Formerly site-local address range. Deprecated in September 2004.
ff00::/8	Multicast	• IPv6 addresses used to communicate with members of multicast groups

information pertaining to network resources, and not address assignment itself. Nodes participating in stateless DHCPv6 must have configured a valid IPv6 address though some other means (Droms, 2004). As a result, this scheme has been deemed irrelevant to the topic.

If DHCPv6 is used to lease addresses from an address pool (a range of addresses that a server is able to provide to clients), the addresses assigned to clients will be spatially close together (i.e. low entropy). The result of doing so reduces the entropy of the

leased addresses, and can, therefore provide a valid precursor to search attempts. This situation can also occur with manual addressing, where an administrator manually assigns an IPv6 address to a host. Whilst it is possible to do so using randomly generated addresses, or high entropy address allocation schemes, the likelihood is that a human is going to address hosts in a memorable fashion. This has been witnessed and cited as a problem in works such as Narten, Draves and Krishnan (2007), where the author makes specific recommendations against the usage of predictable addressing schemes, instead advocating for the use of stochastic methods. Again, the entropy of such an address scheme may be significantly lower than using stochastic address generation methods.

Additionally, due to the standard hexadecimal string representation of IPv6 addresses it is possible to inject words comprised of hexadecimal characters into an IPv6 address. This substitution technique uses the hexadecimal characters `a-f` and the numbers `0-9` to represent words natively, and via hexadecimal word substitutions. For example, the hexadecimal string `0xcafe` can be used to represent the word 'cafe'. An address could be constructed using the hexadecimal strings `0xdead`, `0xbeef`, `0xcafe` and `0xface` to resemble `::dead:beef:cafe:face`. This type of address construction has been observed in real world situations, and has been used to influence search techniques such as those used by THC-IPv6 (Hauser, 2014) and Chiron (Atlasis, 2014). These techniques will be discussed in further detail in Section 2.2.

RFC-4291 (Hinden & Deering, 2006) and RFC-5952 (Kawamura & Kawashima, 2010) discuss the addressing architecture for IPv6. The subsequent changes to the protocol expressed in RFC-5952 (Kawamura & Kawashima, 2010) alleviate some of the concerns with the flexible approach to representing IPv6 addresses in text. Originally there was no formal standard to how leading zeros in IPv6 quartets were represented, and likewise, collapsing consecutive quartets was at the discretion of those using the protocol. In particular, RFC-4291 (Hinden & Deering, 2006) provided discretion on representation with respect to hexadecimal character's case, inclusion of leading zeroes, and address compression using double colon notation. As an example, under RFC-4291 (Hinden & Deering, 2006), the IPv6 address `a001:77:0:77::cafe/64` could be legally represented in, but not limited to, the following ways:

- a001:77:0:77::cafe/64
- a001:0077:0:77::cafe/64
- a001:077:0:77::CAFE/64
- A001:77:0:77::caFE/64
- a001:77:0:77:0:0:0:cafe/64
- A001:0077:0000:0077:0000:0000:0000:cafe/64
- a001:77::77:0000:0000:0000:cafe/64

These loose guidelines presented problems for systems that handled the textual representation of IPv6 addresses. In particular, having loosely defined constraints to displaying and conveying addresses caused problems with superficial parsing systems, such as text-based search and regular expressions (Kawamura & Kawashima, 2010). Due to the number of potential combinations of legal representations of a single address, it was difficult to apply pattern recognition that did not present false positive or false negatives on edge-case addresses. The case of incorrectly handling IPv6 address representations is not dissimilar to the problems that are still occurring with correct handling of email addresses (Klensin, 2001, 2004). RFC-5952 (Kawamura & Kawashima, 2010) enforced stricter control in order to reduce ambiguities with these representations. Referring to the previous example, according to RFC-5952 (Kawamura & Kawashima, 2010), a001:77:0:77::cafe/64 is the only legal string representation of that address. The outcome of Kawamura and Kawashima (2010) is that there are far fewer permutations of possible legal representations of IPv6 addresses.

The string representation of an IPv6 address in a 64 bit subnetwork has character space in the IID for 16 hexadecimal characters. This representation uses colons to separate groups of four hexadecimal characters (see Figure 2-2). The standard IPv4 address representation uses four period separated portions (commonly referred to as dotted-quad or dot-separated decimal notation) containing an integer value between 0 and 255. Another method that could be used as a means by which to construct IIDs is to embed an existing IPv4 address into an IID. The author has named the

approach IPv4-converted-IIDs. IPv4-converted-IIDs can be achieved by substituting the period delimited integer representation into the colon delimited string notation of IPv6 such that the complete IPv4 address becomes the complete IPv6 IID (for example the 32 bit IPv4 address `192.168.1.1` would become the 64 bit IID `::192:168:1:1`). At present no official documentation on this approach has been published. However, it is not inconceivable to expect to observe such addresses in IPv6 implementations, especially throughout the transition stages from IPv4 to IPv6. IPv4-converted-IIDs would simplify the locating of hosts that are IPv4 and IPv6 enabled, since the host address would be effectively the same.

A transition mechanism for injecting IPv4 addresses into IPv6 addresses, named IPv4-mapped-IPv6 addresses, also provides another way to represent addresses. Under this scheme, which is described in RFC-6052 by Bao, Huitema, Bagnulo, Boucadair and Li (2010), the last 32 bits of the address would be replaced with a 32 bit IPv4 address. The string representation also reflected this change, with the last two quartets being changed to the IPv4 period separation scheme. This particular IPv6 address format uses a specific prefix, and does not fall within the global unicast IPv6 address range. IPv4-mapped-IPv6 addresses, therefore, have not been considered during this research. There are also techniques that involve using the full 32 bit IPv4 address as a portion of the 128 bit IPv6 address. With respect to 64 bit networks, RFC-6052 (Bao et al., 2010) defines the following format for embedding IPv4-embedded IPv6 addresses:

1. Place the complete binary IPv4 address from bits 72 to 103 of the IPv6 address.
2. Ensure the bits from 64 to 71 and from 104 to 127 are set to '0'.
3. Prepend the 64 bit network prefix at bits 0 to 63.

An example IPv4 address `192.168.1.1` would be converted into a valid IPv4-embedded IPv6 address with the 64 bit IPv6 network prefix `7777:7:7:7::/64` as follows:

1. Convert IPv4 address `192.168.1.1` into hexadecimal: `c0a80101`.
2. Insert address into bits 72 to 103 of the 128 IPv6 address, keeping the bits 64 to 71 and from 104 to 127 are set to '0': `::c0a8:101:0`.

3. Prepend the 64 bit network prefix `7777:7:7:7::/64` at bits 0 to 63 to complete the address: `7777:7:7:7:c0a8:101::/64`).

IPv4-mapped-IPv6 style addresses were not specifically addressed in this thesis since at the time of writing RFC-6052 (Bao et al., 2010) was still in the proposed standard status. This means that the standard has not been completely ratified, or accepted. If RFC-6052 is ratified the proposed IPv4-mapped-IPv6 addresses could form a valid feature for influencing search heuristics.

In addition to the stateful methods expressed above, host configuration can also be conducted statelessly. The principle method used to statelessly configure IPv6 addresses is through Stateless Address Auto-configuration (SLAAC). SLAAC was defined in Thomson and Narten (1998) and later updated in Thomson, Narten and Jinmei (2007), where it was introduced as a facility for nodes to engage in IPv6 communications without requiring any intervention. SLAAC uses a technique to generate an IID known as Modified EUI-64 (64 bit extended unique identifier - hereafter referred to as EUI-64) which adapts the IEEE MAC-48 (48 bit media access control) address of the node's network interface into a 64 bit IID (IEEE Standards Association, n.d. Narten et al., 2007). EUI-64 defines a standard approach to converting EUI numbers to IIDs. Common EUIs include;

- The IEEE MAC-48 which are used commonly as hardware MAC addresses on network interface cards;
- EUI-48, which are used as identifiers for software products; and
- EUI-64, which are typically used as Ethernet addresses in newer protocols (such as IEEE 802.15.4) (IEEE Standards Association, n.d. Carpene & Woodward, 2012).

Carpene and Woodward (2012) demonstrates the process for creating EUI-64 IIDs using MAC-48 addresses and standard IEEE EUI-64 addresses. The following example conveys how a standard MAC-48 can be converted into a EUI-64 compliant SLAAC IPv6 IID:

1. `00:aa:12:34:56:fe` - Commence with a standard MAC-48 address.

2. `02:aa:12:34:56:fe` - Flip the universal/local bit of MAC-48 (7th bit of the highest order byte).

3. `02aa:12<-->34:56fe` - Split 48 bit address at 24th bit.

4. `02aa:12ff:fe34:56fe` - Pad 16 bits using `1`s, with the least significant bit set to `0` (`0xfffe`) between bits 24 and 40 (Carpene & Woodward, 2012).

An EUI-64 address can be reverse engineered to derive the originating MAC address by flipping the universal/local bit and then removing the `0xfffe` padding. This reverses the EUI-64 process, revealing the original source MAC-48 identifier. The process for generating a modified EUI-64 address out of a standard IEEE EUI-64 address is simpler still:

1. `00aa:1234:56fe:aa56` - Standard EUI-64 address

2. `02aa:1234:56fe:aa56` - Flip the universal/local bit of EUI-64 (7th bit of the highest order byte) (Carpene & Woodward, 2012)

The resulting IID from either EUI-64 conversion method can then be suffixed to a 64 bit network prefix to complete a valid IPv6 address.

Potential privacy issues with the SLAAC IPv6 address generation process were expressed in Groat, Dunlop, Marchany and Tront (2010), Groat, Dunlop, Marchany and Tront (2011) and Dunlop, Groat, Marchany and Tront (2011). Research was conducted into the nature of IPv6 addressing strategies, and how those strategies can be used to identify individual users on a network. Groat et al. (2010) and Groat et al. (2011) concluded in their research that the use of permanent or semi-permanent addressing can violate a user's privacy, since an IID may remain static when crossing different network boundaries. This means that users may be trackable as they participate in IPv6 communications even though their network prefix might differ. Groat et al. (2011) validate this theory by tracking a device throughout a university campus network through its IPv6 IID.

In order to remediate the privacy issues exposed in Groat et al. (2010) and Groat et al. (2011), a mitigation strategy was realised using an IPv6 address proxy system

known as moving target IPv6 defence (MT6D) in Dunlop, Groat, Urbanski, Marchany and Tront (2011). MT6D is a system designed to dynamically allocate host addresses to clients. These addresses are stochastically generated using a shared session key, timestamp and a hashing algorithm (Dunlop, Groat, Urbanski et al., 2011). From an outside perspective, the addresses generated from this method would be indiscernible from the Cryptographically Generated Address defined in RFC-3972 (Aura, 2005) or other stochastically generated addresses. From the perspective of this research, the MT6D address construction mechanism is recognised insofar as they conform to randomly generated address types.

RFC-7217 (Gont, 2014) also provides a method to address privacy issues with the SLAAC generation method described in RFC-2462 (Thomson & Narten, 1998) and RFC-4941 (Narten et al., 2007). The method described by Thomson and Narten (1998) uses the following algorithm to derive a random (but stable) identifier (RID):

$$RID = F(Prefix, Net\_Iface, Network\_ID, DAD\_Counter, secret\_key) \tag{2.1}$$

Where $F()$ is some pseudorandom function, $Prefix$ is the network prefix, Net_Iface is some identifier for the network interface for which the address is being computed, $Network\_ID$ an optional parameter that describes some network identifier (an example is given as the service set identifier (SSID) for 802.11 networks), $DAD\_Counter$ a counter which is initialised as 0 and increments for each time the duplicate address detection mechanism is triggered, and finally, a $secret\_key$ of at least 128 bits that must be initialised as a pseudorandom number (Gont, 2014).

Although Gont (2014) adheres to the recommendation of RFC-4291 (Hinden & Deering, 2006) and expects that any address other than that beginning with binary digits `000` to have a 64 bit network mask, the algorithm allows for arbitrarily sized IID s to be generated by taking $128 - n$ bits, where n is the network size, from the RID to use as the IID. Bits are taken from the RID starting from the least significant bit (Gont, 2014). This results in semi-permanent addresses, that change per network prefix, are persistent and therefore do not expire with each network prefix. This is in contrary to the privacy extensions defined by Narten et al. (2007) in RFC-4941 which

are temporary addresses that are designed to be disposed of after a period interval or conclusion of a network session. According to RFC-7217 (Gont, 2014), this address construction method generates high entropy IIDs.

Other high entropy address construction schemes exist. Most notably are the CGAs mentioned above, that were defined in RFC-3972 (Aura, 2005). These addresses are constructed using the upper order bits of a device's public key. RFC-5535 (Bagnulo, 2009) proposes a standard for Hash-Based Addresses (HBAs), which proposes that IIDs should be constructed through cryptographic hashing. The commonality with the cryptographically generated IIDs and those that are generated through a randomisation function is that they all exhibit high entropy, seemingly random structures. From an outside perspective it would seem as though this style of address would provide resistance or protection from host enumeration strategies. However, it would depend on the seed and algorithm used to generate the address, and with a great enough sample size, a threat actor may be able to predict IID generation.

For the purpose of this research, any address construction types that generate high entropy addresses are classified as stochastically generated. From the perspective of a casual observer, there is no way to categorically determine the exact method of construction of high entropy IIDs, since they appear to be, for all intents, random.

2.2 Host enumeration search algorithms

Host enumeration refers to the locating of networked nodes (i.e. network enabled devices) on computer networks. Enumeration is often undertaken to determine precisely what devices are connected or communicating on a computer network. It is important to understand the search algorithms that underpin common host enumeration strategies. Although host discovery has existed for many years, with the first tools to conduct host discovery dating back to Schemers (2012)'s `fping` in 1992 (currently maintained by Schemers and Schweikert (2014)), there are few search techniques that have been tested. The major search techniques that have historically been used for this purpose are linear searching and randomised searching.

2.2.1 Linear search

The linear search algorithm (otherwise known as sequential search, brute force, or exhaustive search) is a simple and commonly used technique for searching through arbitrary, discrete address spaces. A linear search generally commences from the first item in an address space, and then continues to increment until either the search condition has been met, or the address space is exhausted. A linear search has a worst-case performance of $O(n)$, meaning that it scales linearly depending upon the size of n. For large address spaces, therefore, exhaustive linear scans are inefficient. In such situations heuristics can be introduced in order to reduce the scope of the linear search. This method is used by tools such as Hauser (2006)'s `alive6` to restrict the search scope to target addresses that have a higher probability of being assigned to networked devices.

Linear searching is a popular choice for off-link host discovery, especially against IPv4 networks. Most of the common tools used for host discovery implement some form of linear search (the exception being `masscan` and `ZMap`). A likely explanation for this is due to the simplicity of the algorithm. Linear search algorithms are uncomplicated to implement into computer programs. Also, linear searching generally does not require maintaining large state tables to determine where the algorithm has searched. This information can be inferred by assessing the current address that is being searched. The most notable examples of network enumeration tools that employ linear searching are `nmap`, `fping`, `ARP-scan`, `alive6` and `chiron`. These tools generally provide a means for users to search a range, or multiple ranges of addresses which will be searched sequentially.

Atlasis (2014) introduces an open source tool called the Chiron IPv6 suite containing a program used to scan IPv6 addresses and networks. This scanning program uses two main methods to enumerate IPv6 nodes; first, the program attempts to perform linear searches of user-specified address ranges; second, the program conducts "smart" scanning by testing addresses using permutations of wordy suffixes, such as `face, b00c, beef`, etc. (Atlasis, 2014) in a similar fashion to Hauser (2014) (see Section 2.2.3).

Sequential searches can be combined within a single host enumeration attempt. In such cases an agent may choose to target networks sequentially. Whilst only searching specific host addresses in those networks. This strategy has been termed reverse IP-sequential searching by Leonard and Loguinov (2010). As an example, when performed against some IPv4 networks, an agent may choose to only target addresses one to ten in every standard 24 bit subnetwork (so `*.*.*.1-10`). Such sequential search patterns are offered by tools such as `nmap`. A similar search in IPv6 might resemble `aaaa:1:1:0-ffff::0-ffff`, which would enumerate every host address from `0` to `ffff` in every network from `aaaa:1:1::/64` to `aaaa:1:1:ffff::/64`.

Binary search (also referred to as bisectional search or logarithmic) is an algorithm that can efficiently locate an indexed item in a sorted array of values. Binary search has a worst case performance of $O(\log n)$ and is efficient at searching an ordered list or array for a particular value (Knuth, 1998, pp. 409-417). A cursory consideration would suggest that binary search would be a viable approach to performing IPv6 host enumeration as an alternative to sequential searching, but the nature of the problem defies the algorithm. Binary search techniques are inappropriate for usage in host enumeration problems because the targets are functionally independent. Performing a greater than or less than comparison between functionally independent discrete addresses is not logical. Consequently this algorithm was ruled out for testing in this research, as it isn't suited to the task of host enumeration.

2.2.2 Randomised search

Randomised searching became a desirable strategy once Internet-wide searching became viable. With randomised searching, randomisation algorithms are employed to shuffle the order of the targets in the address space. Performing host enumeration in this fashion has some real-world benefits. Primarily by distributing probes over the entire address space, rather than sequentially probing hosts, the initiator of the host enumeration exercise can avoid overloading target networks that operate sequential resources with probes (Graham, 2013c; Durumeric, Wustrow & Halderman, 2013a), preventing accidental denial of service (DoS) attacks. Likewise, it may also allow the enumeration to remain covert, avoiding detection from intrusion detection systems.

There are also disadvantages to employing randomisation functions to host discovery. A search requiring randomisation is generally going to be more computationally expensive than a linear search. Depending on the randomisation function used, the computational overhead could be significant. Simple randomisation techniques such as encoding values may not add significant overhead. However, employing cryptographic algorithms may. Additionally, with randomisation, maintaining state is an important consideration to prevent duplication of probes, which amount to wasted probes. Again, this problem can be solved algorithmically using permutation-based approaches (see Section 2.2.2.1 or Section 2.2.2.4 for examples of randomised search algorithms that require maintaining minimal state).

Finally, randomisation functions make it difficult to target arbitrary address ranges, or exclude addresses from search attempts. These are also mostly solved problems as well. Excluding addresses from search efforts can be achieved by simply performing a blacklist lookup prior to probing (e.g. checking that the target to be probed does not exist in the blacklist). Targeting arbitrary ranges of addresses is more complicated and must be specifically designed into the algorithm if it is to be achievable.

The coming sections detail the randomisation functions that have been utilised in major host enumeration tools.

2.2.2.1 Generalised-Feistel cipher randomisation algorithm

`masscan` (Graham, 2013b, 2013c) is a network discovery application that is capable of transmitting up to 25 million IPv4 packets per second (pps) on a 10 gigabits per second (gbps) network connection Graham (2013c). Currently `masscan` only supports searching the IPv4 address space, and does not support IPv6. `masscan` gains its performance advantage by using asynchronous sending and receiving of probes, which is to say, it will continuously send a probe (e.g. TCP SYN segments for a half-open scan) and handle the responses (e.g. the TCP SYN/ACK segments) as they arrive. This is contrary to how applications such as `nmap` operate, in that they will send a probe and wait for a response, or the appropriate timeout period before sending the next probe.

In addition to asynchronous probing, `masscan` gains a performance improvement over other utilities (such as `nmap`) by bypassing the underlying kernel of the operating

system, and delivering packets directly to the network hardware. Part of the success of `masscan` comes from its reimplementation of the IP and TCP protocol stacks, from within the application, rather than relying on the OS implementation of these protocols. By operating entirely through raw sockets, the application is able to reduce the overhead used by these protocols by processing only packets relevant to the search exercise and ignoring other superfluous data streams. These techniques aren't necessarily relevant to the research topic, but may influence future research strategies used for conducting host enumeration against live IPv6 networks.

Of particular interest and relevant to this thesis is `masscan`'s (Graham, 2013c) randomisation function. In order to reduce the load on target networks, `masscan` performs an exhaustive search using a randomisation function. The function `masscan` uses to randomise the order of target IPv4 addresses is based upon the cryptographic 'Generalised-Feistel Cipher' detailed in Black and Rogaway (2002). The modified version of the Generalised-Feistel cipher that was used in Graham (2013b) has been referred to as the "BlackRock" algorithm. The Generalised-Feistel cipher applies a form of encryption to indices in a range of addresses, and then applies a modulus operation to ensure that the result is within the bounds of the address space.

The Generalised-Feistel cipher is broken into two separate algorithms; the encryption (encipher) algorithm, which is depicted in Algorithm 1 and the decryption (decipher) algorithm, which is displayed in Algorithm 2. In Graham (2013b), the author chooses to use a modified data encryption standard (DES) operation as the cryptographic algorithm.

The algorithm works by performing a sequential search of the address space, but rather than probing each address, an encryption function, outlined in Black and Rogaway (2002) (see Algorithm 1), is applied. A set of numbers (M) between 0 and k represents the number of targets to be probed. For each target address (m) in M, the encryption function is applied. The resulting randomised address (c) is then checked to ensure it exists within the address space that is being searched. If it does not exist within the appropriate address space, the iteration continues. Otherwise the resulting encrypted address (c), is treated as the target for the search operation, and probing is conducted. In addition, this operation provides a cost effective method of resuming a search op-

Algorithm $Fe[r, a, b]_K(m)$*;*
c $\leftarrow fe[r, a, b]_K(m)$;
if $c \in M$ **then**
  return c;
else
  return $Fe[r, a, b]_K(c)$;
end
Algorithm $fe[r, a, b]_K(m)$*;*
L $\leftarrow m \mod a$;
R $\leftarrow m/a$;
for $j \leftarrow 1$ **to** r **do**
  if *(j is odd)* **then**
    tmp $\leftarrow (L + F_j(R)) \mod a$;
  else
    tmp $\leftarrow (L + F_j(R)) \mod b$;
  end
end
L $\leftarrow$ R;
R $\leftarrow$ tmp;
if *(r is odd)* **then**
  return $aL + R$*;*
else
  return $aR + L$*;*
end

Algorithm 1: Generalised-Feistel Cipher adapted from Black and Rogaway (2002). For this cipher ($Fe[r, a, b]$), m represents a set $[0, k-1]$, r is the number of rounds used within the Feistel network, and a and b are positive numbers such that $ab \geqq k$ (Black & Rogaway, 2002). These functions implement the encipher operation on the data.

eration, provided that the last searched address is recorded along with the encryption key. Resumption can be then accomplished by performing the deciphering operation described in Black and Rogaway (2002) (see Algorithm 2). This operation will return the index value that was used as the last target.

2.2.2.2 Linear congruential generator randomisation algorithm

A linear congruential generator (LCG) is a method that can be used to generate pseudorandom numbers in pseudorandom number generators (PRNG). LCGs have the potential to be full cycle algorithms, which is to say that they can generate every valid discrete number between 0 and n, where n is the size of the generator.

`nmap` (G. F. Lyon, 2009) uses an LCG to create a range of target IP addresses, in order to randomise the order of probing. This randomisation function is executed

Algorithm $Fe[r, a, b]_K^{-1}(m)$*;*
c $\leftarrow fe[r, a, b]_K^{-1}(m)$;
if $c \in M$ **then**
  return c;
else
  return $Fe[r, a, b]_K^{-1}(c)$;
end
Algorithm $fe[r, a, b]_K^{-1}(m)$*;*
if *(r is odd)* **then**
  R$\leftarrow m \mod a$;
  L$\leftarrow m/a$;
else
  L$\leftarrow m \mod a$;
  R$\leftarrow m/a$;
end
for $j \leftarrow r$ **to** 1 **do**
  if *(j is odd)* **then**
    tmp $\leftarrow (L - F_j(R)) \mod a$;
  else
    tmp $\leftarrow (L - F_j(R)) \mod b$;
  end
end
R $\leftarrow$ L;
L $\leftarrow$ tmp;
return $aR + L$*;*

Algorithm 2: Generalised-Feistel Cipher adapted from Black and Rogaway (2002). For this cipher ($Fe[r, a, b]$), m represents a set $[0, k-1]$, r is the number of rounds used within the Feistel network, and a and b are positive numbers such that $ab \geqq k$ (Black & Rogaway, 2002). These functions implement the decipher operation on the data.

when the program is invoked with the *-iR* switch. This LCG (which is visible in the program's source code in the file `nbase/nbase_rnd.c` (G. Lyon, 2015)) has been adapted and is included in Algorithm 3. The magic numbers used within the source code are explained to be constants from the Numeric Recipes book, as well as constants from `glibc` and `Quick C/C++` (G. Lyon, 2015). The algorithm only requires that five variables are persistent between calls to the function in order to complete the LCG cycle. This enables the algorithm to resume state between runtimes, provided the $State, Tweak1, Tweak2, Tweak3$ variables are recorded, and the $StateInit$ flag is set to a boolean True value (i.e. 1).

A similar technique was used in the Witty worm's propagation which affected over 12,000 computers in 75 minutes in 2004 (Kumar, Paxson & Weaver, 2005). According

Data: State, Tweak1, Tweak2, Tweak3
Result: A random unsigned integer $0 \leq n < 2^{32}$
$StaticStateInit \leftarrow 0$;
`// If State hasn't been initialised, initialise it to a random 32 bit integer, along with the other tweak modifiers.`
if *!StateInit* **then**
  $State \leftarrow Rand(0, 2^{32})$;
  $Tweak1 \leftarrow Rand(0, 2^{32})$;
  $Tweak2 \leftarrow Rand(0, 2^{32})$;
  $Tweak3 \leftarrow Rand(0, 2^{32})$;
  `// Set flag to ensure this routine is not called again`
  $StateInit \leftarrow 1$;
end
$Output \leftarrow State$;
$State \leftarrow (((State * 1664525)\&0xFFFFFFFF) + 1013904223)\&0xFFFFFFFF$;
`// Round 1: rotate Output and XOR against Tweak1 modifier`
$Output \leftarrow ((Output << 7) | (Output >> (32 - 7)))$;
$Output \leftarrow Output \otimes Tweak1$;
`// Round 2: rotate Output, subject it to an affine transform and finally XOR against Tweak2 modifier`
$Output \leftarrow$
$(((Output * 1103515245)\&0xFFFFFFFF) + 12345)\&0xFFFFFFFF$;
$Output \leftarrow ((Output << 15) | (Output >> (32 - 15)))$;
$Output \leftarrow Output \otimes Tweak2$;
`// Round 3: rotate Output and XOR against Tweak3 modifier`
$Output \leftarrow (((Output * 214013)\&0xFFFFFFFF) + 2531011)\&0xFFFFFFFF$;
$Output \leftarrow ((Output << 5) | (Output >> (32 - 5)))$;
$Output \leftarrow Output \otimes Tweak3$;
return *Output;*

Algorithm 3: The linear congruent generator used in the `nmap` program (specifically in the file `nbase/nbase_rnd.c`) (G. Lyon, 2015; G. F. Lyon, 2009). The randomisation function makes use of magic numbers to ensure that output is random, unique and that the LCG comes full cycle.

to Kumar et al. (2005), Shannon and Moore (2004) and Graham (2012) this randomisation technique contained a fundamental flaw in the way the LCG was used. The error was introduced when, rather than generating a random number using 32 bits, which would be required to guarantee the LCG would permute the entire IPv4 address space, the authors of Witty performed a concatenation operation on two 16 bit integers. The two 16 bit integers were acquired by taking the most significant 16 bits returned from a PRNG function. The flaw is evident in Step 3 of the derived pseudocode for the reverse engineered program (Kumar et al., 2005), reproduced in Equation 2.2.

$$3.dest\_ip \leftarrow rand()_{[0\cdots15]}||rand()_{[0\cdots15]}; \tag{2.2}$$

By generating the randomised target IP address thusly, the authors inadvertently reduced the address space to be targeted, and introduced duplication to the algorithm. It can be seen from this example that care must be taken when designing and implementing randomisation functions to ensure that the algorithm functions correctly.

2.2.2.3 RC4 cipher randomisation algorithm

`nmap` also utilises another randomisation function which involves an RC4 cipher when the program is executed with the *–randomize-hosts* switch. The fourth Rivest cipher (RC4), named after its creator Ronald L. Rivest (Rivest & Schuldt, 2014), is a cryptographic stream cipher that is typically used to encrypt and decrypt arbitrary data streams.

With this particular implementation, an array is shuffled in memory using the cryptographic cipher, altering the positions of the elements. The algorithm achieves a randomised distribution of the target addresses stored in the array. The execution of this approach in `nmap` has limitations, since the randomised array of IP addresses is maintained in resident memory throughout the runtime of the application. According to Graham (2013a) the function that is used only allows the checking of 16,384 hosts at a time. This approach has been criticised by Graham (2013a) and Graham (2013c) as being inflexible and memory intensive. According to G. Lyon (2015) this algorithm is also subject to the 'Birthday Paradox' which means it is susceptible to producing duplicate addresses and that the probability of a duplication will increase with every address checked.

2.2.2.4 ZMap's randomisation algorithm

`ZMap` (Durumeric et al., 2013a, 2013b) is another high performance host discovery program. Similarly to `masscan`, `ZMap` currently only supports searching the IPv4 protocol's address space. To gain a performance advantage `ZMap` uses raw sockets for probe generation and delivery. Using raw sockets is preferred in this case because it allows `ZMap`

to bypass the OS kernel, and prevents unnecessary lookups and checks from being conducted. Also similarly to `masscan`, `ZMap` relies upon a randomisation function to determine the order of address probing. The function `ZMap` uses relies upon the nature of multiplicative groups and modulus arithmetic operations to achieve an exhaustive, deterministic, non-sequential and cyclical permutation of the target address space. In effect this means that the algorithm will generate a target address list that covers every address in the 32 bit target address space, except address 0 (ip address `0.0.0.0`), in a pseudorandom fashion.

`ZMap's` randomisation algorithm can be deconstructed into the following stages:

1. Choose a prime number, slightly larger than the address space. In the case of Durumeric et al. (2013a) the authors pick $p = 2^{32} + 15$ as the first prime number above 2^{32} (which represents the size of the IPv4 address space).

2. Calculate and select a primitive root for the prime number p using a method such as that which is described in Weisstein (n.d.).

3. Begin with a random address in the address space. Starting from a random point in the address space is not essential. However, when initialised with different primitive roots, a randomised starting position allows for different sequence permutations of the address space.

4. If the target address is less than the size of the address space, then probe the target address.

5. Update the target such that $Target = (Target * PrimitiveRoot) \mod p$.

6. If the updated target address is equal to the first address that was tested, then the algorithm has completed a cycle and can be concluded, or the cycle can commence again.

A modified version of the algorithm used in Durumeric et al. (2013a) is included in Algorithm 4.

This algorithm is less computationally expensive than the Generalised Feistel algorithm described in Graham (2013b), Black and Rogaway (2002). The ZMap randomisation algorithm performed six times faster than the Feistel algorithm in a sample

Data: AddressRange, PrimitiveRoot, p, FirstTarget
Result: A cyclical, deterministic, non-sequential permutation of the address space to be searched

$AddressRange \leftarrow (2^{64}) - 1$;
// AddressRange refers to the end address in the target address space that is the subject of the search effort. E.g. if it is intended to search between address 0 and address 2^{64} exclusive, the AddressRange would equal 2^{64}
$p \leftarrow NextPrime(AddressRange)$;
$PrimitiveRoot \leftarrow GetPrimitiveRoots(p)$;
$FirstTarget \leftarrow Rand(1, AddressRange)$;
$Target \leftarrow FirstTarget$;
while *True* **do**
  if $Target \leq AddressRange$ **then**
    Probe(Target);
  end
  $Target \leftarrow ((Target * PrimitiveRoot) \mod p)$;
  if $Target == FirstTarget$ **then**
    Break;
  end
end

Algorithm 4: Adaptation of the algorithm used for the cyclical permutation algorithm in Durumeric, Wustrow and Halderman (2013a). The algorithm provides a means to randomly traverse a target address space without probing any addresses more than a single time.

test conducted by the author, using Python-based implementations of each algorithm. In these tests it was determined that 1000 address mutations were performed in 0.005 and 0.030 seconds for the ZMap algorithm and the Generalised Feistel algorithm respectively. Similarly to the LCG and the Generalised Feistel algorithm, the algorithm also provides a means to resume search operations that requires recording only the $FirstTarget$, $Target$, p and $PrimitiveRoot$ variables.

2.2.3 IPv6 IID pattern search techniques

The common hexadecimal string representation of IPv6 addresses introduces a vector for potential patterns to occur in addresses. As mentioned in Section 8, the hexadecimal characters `0-9` and `a-f` can be manipulated to create natural language words. For example, the hexadecimal words `0xbeef` and `0xb00c` can be used to resemble the words 'beef' and 'book' respectively. It has been noted that these patterns are actively being used in address construction in real-world IPv6 deployments. Specifically these

patterns were identified in Carpene and Woodward (2012), where it was shown that organisations were using these hexadecimal word substitutions in their IPv6 addressing scheme.

The Hacker's Choice IPv6 toolkit (THC-IPv6) is an application suite geared towards malicious usage of the IPv6 protocol stack. In this suite a utility called `alive6` can be used to search an IPv6 network (Hauser, 2006). `alive6` uses a combination of searching common hexadecimal word substitution patterns and low range incrementation of IIDs to significantly reduce the target address space. Also, as previously mentioned, in conjunction with a linear search, `chiron` (Atlasis, 2014) also performs a 'smart scan' whereby it will attempt to address devices using hexadecimal word substitutions. The search mode will take a list of hexadecimal 'word' strings and perform a multi-choose to construct addresses using them. For example, if the hexadecimal words `face, b00c, beef, cafe` were included it would generate IPv6 addresses of all 256 combinations of the hexadecimal words as partially depicted below:

- `::face:face:face:face`
- `::face:face:face:b00c`
- `::face:face:face:beef`
- `::face:face:face:cafe`
- `::face:face:b00c:face`
- `::face:face:b00c:b00c`
- ...
- `::cafe:cafe:cafe:beef`
- `::cafe:cafe:cafe:cafe`

These addresses form the target list for the search attempt, and are then exhausted sequentially. This approach to searching has not undergone rigorous testing, but shows promise as a heuristic search method for IPv6.

2.3 Host enumeration methods

With an understanding of the common search strategies used within host enumeration strategies, discussion can move to focus on the methods themselves. Host enumeration methods can be classified into two major categories; active enumeration and passive enumeration. Active enumeration (or direct enumeration) involves the explicit probing of target systems, and recording responses to enumerate devices. Examples of strategies used for direct enumeration include ping sweeps, ARP scans, Neighbour Discovery Protocol scanning, TCP SYN scanning, etc.

Passive enumeration (or indirect enumeration) involves enumerating devices through means that do not require the direct probing of target systems. As an example, using passive reconnaissance strategies to listen for network communications between devices, or through querying Domain Name System (DNS) or SNMP about potential hosts. This research is primarily focused on active enumeration. However, for completeness, passive enumeration methods are also discussed.

The two categories of enumeration, along with the components of each, and common methods used are included in Table 2.2.

Table 2.2: A dissection of the major categories of host enumeration, active and passive enumeration, including the components and common methods used for each.

	Active enumeration	Passive enumeration
Description	Probes target system(s) directly, recording responses to enumerate devices.	Harvests information from ancillary sources to enumerate devices
Components	• Target address space • Probe target • Search algorithm • Protocol • Payload	• Reconnaissance subject • Reconnaissance object • Protocol • Payload
Major methods	ARP scanning, ICMP echo probing, ICMP NIQs, TCP port scanning	Passive network monitoring, Website scraping, DNS enumeration, SNMP querying, Reconnaissance from network services

2.3.1 Active enumeration methods

A framework describing the components of active host enumeration exercises is, thus far, absent from the literature. This thesis offers a five component model for classifying

active host enumeration methods. These five components to conducting active host enumeration exercise are:

1. The target address space. The target address space is the range (or ranges) of addresses that are valid targets for the host enumeration exercise. In the case of this research, the target address space is a single 64 bit IPv6 subnetwork (i.e. the network `::/64`). In other host discovery efforts, such as those against IPv4 networks, the target address space may be different. Note that the target address space determines whether the effort is on-link or off-link.

2. Probe targets. Elements in the target address space range. Not to be confused with the target address space, this refers to the delivery address of probes. The distinction is made, since the target address may differ from the target address space in cases that involve leveraging special addresses (such as the on-link methods discussed below).

3. The search algorithm. The search algorithm is the feature of the host enumeration exercise that determines the order in which targets are probed. The search algorithm can be deterministic in nature or stochastic. The commonly used algorithms, such as linear searching and randomised searching, were discussed in Section 2.2. Algorithms with potential for the task that have yet to be tested will be discussed in Section 2.3.4.

4. The protocol. The protocol relates to the actual protocol used to deliver the probe and receipt a response. Examples of protocols for probing include ICMP Echo Requests, TCP SYN segments, ARP requests, etc.

5. The probe payload. Probe payload refers to the contents of the probe. In some cases a generic payload may be used, in other cases specifically crafted payloads may be used. The payload can be used to test for specific vulnerabilities in services, or trigger specific responses from devices to ascertain whether a host is alive. Examples of payloads include randomised data or specifically crafted HTTP GET requests.

2.3.1.1 Address resolution protocol scanning

As previously mentioned, on-link host enumeration refers to host discovery that is performed on a single link layer network and broadcast domain. Consequently, on-link strategies have access to techniques that are not available to off-link actors. Whilst it is feasible for on-link actors to utilise off-link strategies as well, there are generally more efficient options.

ARP is a protocol that resides at layer 1 of the TCP/IP model. It is used to map link layer addresses to network layer addresses for communications between devices participating in IPv4. Since link layer broadcast addresses instruct link layer networking equipment (such as network switches) to deliver frames to all connected nodes within a broadcast domain, this facility can be leveraged to deliver probes to all devices on a network.

An ARP ping is an example of a technique that leverages link layer broadcasts. Typically this method will involve a device broadcasting an ARP request query for a target network layer address. A device who owns the requested IP address will reply to the query with their MAC address. These replies can then be recorded to enumerate devices. By sending consecutive ARP requests that exhaust every possible IP address on the network, an agent can enumerate devices. The ARP ping strategy is employed in the tool `arping` (Habets, 2009).

2.3.1.2 Neighbour discovery protocol scanning

NDP replaces ARP in IPv6 and provides similar facilities for determining alive nodes on a local network, using ICMPv6 to transfer messages. Similarly to ARP, an IPv6 enabled device can solicit other devices on a network to determine who has a particular address. In IPv6 this is accomplished through the use of ICMPv6 messages.

In particular neighbour solicitation (NS) requests can be sent by nodes to the link-local `all-nodes` multicast group (`ff02::1`). IPv6-enabled devices will then respond with their link-layer address. This enumeration method is one of the common methods that the literature suggests using for on-link IPv6 host enumeration, and is utilised in the `chiron` suite (Atlasis, 2014), `nmap` (G. F. Lyon, 2009), and THC-IPv6 suite's `alive6`

(Hauser, 2014). There are also other multicast group addresses used by NDP that are interesting for an agent to probe, such as the `all-routers` address (`ff02::2`), or the `all-dhcp-agents` address (`ff02::1:2`). A full taxonomy of the link-local multicast groups reserved by IANA for IPv6 is included in IANA (2015).

2.3.1.3 ICMP echo probing

ICMP defines a standard for probing hosts using echoes. Except where relevant to differentiate, ICMP will refer to both the ICMPv6 protocol and the Internet Control Message Protocol version 4 (ICMPv4) protocol hereafter. By employing this strategy, a device can send another device an echo request, and wait for the corresponding echo reply. If the response is received in a timely fashion the device is said to be active. Applications, such as the `ping` and `traceroute` utilities, employ this strategy to assist in diagnostic processes. It should be noted that in real-world situations ICMP echo probing is a relatively naive method of determining a node's status. ICMP echo probing is considered to be unreliable due to the fact that, in some cases, ICMP packets are outright denied by network policies which result in false negatives. Network administrators may choose to filter ICMP traffic as a security through obscurity measure in an attempt to hide devices on their network. Additionally, edge or perimeter devices, such as routers and firewalls, might respond to ICMP echoes on behalf of devices on their network, thus creating false positives. This technique is another example of security through obscurity. Finally ICMP might be rate-limited on any intermediary device between the source and destination network nodes, providing erroneous latency results, or causing erroneous timeouts.

One common host discovery technique that uses ICMP is the ping sweep. A ping sweep involves sending ICMP Echo requests to a range of IP address on a network segment and recording the responses. The ICMP ping sweep (amongst other techniques) is used in the network mapping utility `nmap` (G. F. Lyon, 2009). Another common method for on-link discovery involves the network layer IPv4 broadcast address. In IPv4 the last IP address in a network (where the host bits are all set to 1) is a reserved address known as the broadcast address. The broadcast address delivers packets to all addresses on a subnetwork. By sending a request to the broadcast address there is the

possibility that a number of, if not all of the devices on an IPv4 network will reply.

As previously mentioned, in IPv6, the `all-nodes` group includes all IPv6 enabled nodes on a network. An agent can exploit this by sending probes to the multicast group address, using for example ICMPv6 echo requests, to which each node should respond with an echo reply. There are also other standard groups for link-local devices; the `all-routers`, which can be probed in a similar fashion. These methods are utilised in tools such as THC-IPv6's `alive6`, and `nmap` to conduct on-link enumeration of IPv6 network nodes, and are the most efficient means of enumeration for on-link devices.

2.3.1.4 ICMP node information queries (NIQs)

Additionally, ICMPv6 includes IPv6 NIQs which are defined in RFC-4620 (Crawford & Haberman, 2006)) that allow on-link IPv6 devices to discover their neighbours. The protocol could be extended to respond to public NIQs. However, it is explicitly stated by Crawford and Haberman (2006) that this functionality should be disabled by default. The recommendation is made due to the security implications of exposing a potential reconnaissance interface to public IPv6 networks. The complexity of host enumeration efforts would be reduced if NIQ were permitted from public networks. As an example, an agent could send specifically crafted queries to routing devices and record the responses of connected devices.

2.3.1.5 TCP port probing

Another technique commonly used to determine whether or not a node is alive on a network is through service probing. Service probing refers to the act of initiating a TCP connection (using a SYN segment) with an IPv4/6 address and a port number and listening for SYN/ACK segments. By querying commonly used service ports (such as the HTTP port 80, or the HTTPS port 443), or ranges of ports on a particular host (e.g. ports 1 to 1024), an actor can increase the likelihood of discovering a live device (Kim & Solomon, 2010).

Completing a TCP three-way handshake is not required for an agent to determine whether a port is open or not. As a result, the final stage of the three-way handshake (returning an ACK segment) is generally omitted. This strategy is known as SYN

scanning or half-open scanning (Kim & Solomon, 2010). TCP port probing offers an alternative to ICMP ping scanning when a node doesn't respond to ping requests, or where the intention of the enumeration attempt is to determine whether or not devices are listening for certain services. TCP SYN probing has been successfully used in research projects, such as those detailed in Section 2.4.2. It is commonplace for host enumeration and VA tools to provide a facility to conduct TCP port probing. Applications such as `nmap`, `ZMap`, `masscan`, and `nessus` allow users to perform TCP port probing as a probing mechanism.

An agent can also perform enumeration of TCP ports on a host in an effort to determine the ports it has open or is listening on. By probing a series of ports on a single host an agent can enumerate all of the TCP ports that are listening for connection attempts. TCP port enumeration, whilst outside the scope of this research, is an important component of vulnerability assessments.

2.3.2 Passive enumeration methods

Passive enumeration methods are those that do not probe the target systems themselves, but rather employ other means to enumerate devices. This literature review has identified four major components to passive enumeration strategies. These components are:

1. The reconnaissance subject. The reconnaissance subject is the systems that are being enumerated for information. The reconnaissance subject could be a network, IP address, domain name, website etc. As an example, for this research the reconnaissance subject is the IPv6 address space `::/64`.

2. The reconnaissance object. The reconnaissance object is the device or system that is being interacted with to gain information about the reconnaissance subject. This could be an Ethernet switch on a computer network, a DNS server, a website, etc.

3. The protocol. The protocol relates to the actual protocol that is used or examined to gather information about the reconnaissance subject. The protocol could be DNS, SNMP, HTTP, or Ethernet frames or IP(v4/v6) packets.

4. Payload. The payload is the data that is either sent or examined from communications to gather information. For example, in a DNS enumeration, the payload may be a list of subdomains in a query. The response payload may be the answers from the DNS server (the A or AAAA record depending on the reconnaissance subject). In a passive network monitoring enumeration, the payload would be the data held within packets that are interesting to the observer.

Generally, passive enumerations strategies rely on utilising third party systems. For example, by querying DNS servers for potential subdomains, or by passively monitoring network transmissions. These strategies can be used to enumerate more than just device IP addresses or available services. Zalewski (2005) postulates that passive enumeration could be employed to enumerate subdomains, MAC addresses, website resources, interface names/numbers etc. According to Glassman and Kang (2012), passive enumeration that uses only publicly available information sources to enumerate information about a target is referred to as open source intelligence (OSINT).

2.3.2.1 Domain name system enumeration

DNS is used to map human readable names to computer and network resources. That is to say that a DNS server is used to convert domain names to IP(v4/v6) addresses. DNS uses a server-client model to provide name resolution. Under the system, a client that knows a particular domain name for an Internet resource can query a server which will conduct a series of lookups (potentially involving referring the request to other servers) and return the IP number associated with that name resource. The client can then use the IP number to make a network connection as necessary.

This name resolution mechanism can be leveraged to enumerate network resources. If an organisation's network devices have been named and published to public DNS, an agent can query the public DNS for potential host records and record the responses. The DNS enumeration strategy is made less complex when administrators use predictable subdomain names for their network resources (e.g. `www.domain.name` for a website or `mail.domain.name` for a mail server).

Another method of conducting DNS enumeration relies on requesting the zone file

from a DNS server in the form of a zone transfer. DNS zones are files on a DNS server that contain the DNS records (including A records for IPv4, AAAA records for IPv6, TXT records, SRV records, etc.) for all known members of a domain. DNS allows clients to request copies of DNS zones from servers, which, if successful, allows clients to copy all of the DNS records a server contains about a domain.

2.3.2.2 SNMP

The simple network management protocol (SNMP) is a protocol that is used to monitor and manage networking equipment in a computer network. SNMP provides an interface for administrators to request information from devices and also issue commands to devices. SNMP can be leveraged to enumerate devices by querying SNMP-enabled network devices for information such as the device's ARP table, which contains the known IPv4 address to MAC address mappings. For IPv6, an agent could query a device for the items contained in the Table ipv6IfTable (registered OID `1.3.6.1.2.1.55.1.5`) which contains information about a device's IPv6 interfaces, including their addresses (Net-SNMP, 2011). Furthermore, the items contained in Table ipv6NetToMediaTable (registered OID `1.3.6.1.2.1.55.1.12`) include information about network addresses to link layer addresses for IPv6 (Net-SNMP, 2011).

2.3.2.3 Website enumeration

If an organisation has a website, information can be gathered from the website to determine other nodes on the organisation's network. As an example, by extracting any universal resource locators (URLs) from a company's website, an agent might discover other subdomains or domains belonging to the company. Subdomain information can then be used in a DNS enumeration to obtain the IP numbers for the resources. Likewise, any exposed database connections, or asset locations on a website could provide further information for the agent conducting the enumeration.

2.3.2.4 Passive network monitoring

Monitoring network transmissions for unique host addresses is another form of passive host enumeration. Passive network monitoring may be used by network administrators

who are performing census style surveys on their network usage, rather than actors who wish to determine the live nodes on their networks. Polcák (2014) discusses the use of passive means to gather information about unique hosts, and will be further detailed in 2.4.3.

On-link network enumeration can be conducted by passively monitoring network transmissions on a broadcast domain. Unique IP address details can be collected from the payload of network transmissions (such as from ARP requests), or from the source or destination addresses of communications. Passive monitoring is an appropriate enumeration strategy against IPv4 networks. Broadcast packets, which are delivered to all nodes in a broadcast domain, are utilised by IPv4. IPv6 is more complicated. With IPv6, heavy usage is made of multicast groups, since the protocol does not allow broadcasting. Subsequently, less traffic is generated that is intended to reach all clients. In any case, passive network monitoring is still a viable method of passive enumeration against IPv6 networks.

Alternatively, an agent may also be able to infer nodes on a network by passively monitoring the network communications that ingress and egress a network segment. These network communications will contain a source and destination address corresponding to a device on the local network and potentially an off-link device or another on-link device. By recording the unique addresses observed, an agent can eventually determine the live devices on a network.

A functional network requires a variety of protocols to interoperate in order to facilitate efficient communications. Services such as DHCP and DNS can not only provide network resources to client devices, but also act as potential information sources to aid in enumeration attempts. As an example, DHCP is used to provide network resources (including network address configuration) to clients. These messages may be listened to by an agent to discover which addresses have been allocated to clients. Protocols such as the Cisco discovery protocol (CDP), network time protocol (NTP), etc., may also contain information about host devices within their payload, whether that be the addresses of servers or clients. An agent that can eavesdrop on these messages for a period of time may be able to infer a large amount of information about the network design and the active nodes within it.

This enumeration strategy requires the actor to obtain a network vantage point that positions them between the network of interest (i.e. the reconnaissance subject) and an egress point. A vantage point could be acquired by connecting a wire tap, an ethernet hub, or a computer, to the computer network, or it could involve hijacking an intermediary device on the network. Specialised devices are also available that can perform passive network monitoring for the purpose of host discovery and identification, vulnerability monitoring and networked device profiles and usage baselines.

2.3.2.5 Open source intelligence (OSINT)

Public information repositories also exist to catalogue website addresses, vulnerable hosts on the Internet, and even some popular IPv4/IPv6 addresses. These online services can also be used as a means to garner information. Some examples of such services include the passive DNS project, WHOIS, and ShodanHQ. Toolkits such as `Maltego` can be used to gather open source information from a variety of sources, in order to enumerate network devices, as well as other information about a reconnaissance subject (such as email addresses, phone numbers, websites, other network services, etc.).

2.3.3 Comparison between IPv4 and IPv6 off-link host enumeration

At present there exists a number of methods by which to enumerate hosts within an off-link IPv4 network, such as those described in Graham (2013c), Durumeric et al. (2013a), and G. F. Lyon (2009). Leonard and Loguinov (2010) also discuss many of the common search methods used for host enumeration in IPv4 and identify the linear search, randomised search, and the reverse ip-sequential search as methods used to enumerate IPv4 networks.

Of the methods available to IPv4 host enumeration, the literature suggests that the linear search has been applied to the IPv6 realm in Hauser (2006) and Atlasis (2014), but has yet to be validated. In addition to the linear search, IPv6 tools are available that abuse the nature of hexadecimal numbers to search possible IPv6 addresses that may contain hexadecimal words. Little other literature exists that addresses methods that may apply to off-link IPv6 host enumeration, and the literature that is available

suggests that such efforts are futile.

These claims are discussed in Chown (2008) and Durumeric et al. (2013a). Chown (2008) proposes that since the address space for IPv6 is so vast, it is infeasible to attempt host enumeration. Durumeric et al. (2013a), elaborate further, and make specific mention of the notion that IPv6 searching methodologies are not currently available or mature enough to facilitate Internet-wide scanning against the IPv6 protocol. The authors make the assumption that the strategies used in `ZMap` to enumerate the IPv4 address space would not suffice when applied to the IPv6 address space.

Leonard and Loguinov (2010) also conducted research into finding the most efficient delivery methods for performing enumerations across the entire 32 bit address space. With this study, the authors take into consideration known information about the allocation of IPv4 addresses by the governing number allocation authority (IANA) published in IANA (2014a). This allows them to eliminate areas of the address space that are either unallocated or unused for public purposes, and create a subset of the address space that encompasses the most appropriate target addresses. Since IPv4 does not provide the same clear distinction between the host portion of an IP address and the network portion that IPv6 does, any host in the remaining subset is a potential valid target. Leonard and Loguinov (2010) address two of the major problems with host enumeration techniques; reducing the scope of the address space; and choosing the transport medium for probes.

This study differentiates significantly from the research conducted by Leonard and Loguinov (2010) since this research is concerned with the host portion of the IPv6 address space. The IPv6 address space is not governed by the same address assignment policies as IPv4 is. Consequently, in a single 64 bit subnet, any discrete address could be a valid host. If the research was concerned with enumeration of IPv6 networks, or the entire IPv6 Internet, Leonard and Loguinov (2010)'s work provides a valid foundation for conducting the research. Secondly this research is not concerned with the transmission mechanism for delivering probes. This research aims to provide methods for performing host enumeration or searching large discrete spaces. The protocol or payload used to probe or test hosts is not considered within the scope of the research.

It is apparent from the available literature on the topic that conventional approaches

are not appropriate for IPv6 host enumeration. Exhaustive searching (of either subnetworks or the entire address space), as is popular and commonplace with IPv4 host enumeration tools and strategies, is simply not possible with current computing resources. Any strategies that have been developed to address the problem must consider this limitation. Hauser (2014) and Atlasis (2014) accepted this limitation, and reduced the scope of their search efforts through searching a subset of the address space. Translating IPv4 host enumeration strategies provides one such means of addressing the problem. However, perhaps a different approach and perspective is required. Machine learning systems may provide some recourse for IPv6 host enumeration strategies. Thus far, learning or decision making systems have not been tested in this problem domain.

2.3.4 Machine learning and host enumeration

Machine learning systems refer to systems that learn from input data, and use this acquired knowledge to make decisions about supplementary data. Machine learning can be achieved using either supervised or unsupervised learning methods (scikit-learn developers, n.d.). Under supervised learning a machine learning system is provided with a sample of pre-classified training data, which is used to train the learning system. The system can then be used to test against unclassified data. Unsupervised learning by contrast, attempts to discover features in an unknown dataset, and does not require training (Hastie, Tibshirani & Friedman, 2013).

There are four main problem domains that machine learning systems seek to assist with; classification, regression, clustering and dimensionality reduction (Hastie et al., 2013). Classification problems involve attempting to classify data into categories known in advance by features. Classification problems require supervised learning strategies, and the outputs are discrete values. Regression problems also require supervised learning strategies. Regression strategies do not output a discrete value, but rather a continuous value. Clustering problems are akin to classification problems in that they produce discrete outputs, except that they utilise unsupervised learning techniques, and therefore do not require labelled data as inputs. Clustering methods attempt to group data based on common features. Dimensionality reduction is another unsupervised learning strategy that seeks to reduce the features of input data

to a smaller set of artificial features that retains most of the original data's variance (scikit-learn developers, n.d.). Dimensionality reduction can be used for exploratory research into datasets, in order to determine strong or useful features as candidates for the independent variables of other machine learning projects. Table 2.3 shows the four main categories of machine learning system.

Table 2.3: Comparison of four main categories of machine learning, their output types, their learning process, and their major purposes.

Machine learning category	Output type	Learning type	Purpose
Classification	Discrete class label(s)	Supervised	Classifying data into known classes, or group membership. Often used in systems that classify data, such as species testing, email spam classifiers, etc. (Hastie, Tibshirani & Friedman, 2013).
Regression	Continuous value(s)	Supervised	Approximate a response from data or make a prediction about data. Often used in prediction systems, such as weather, stock prices, age testing, etc.
Clustering	Discrete group(s)	Unsupervised	Clusters features together into categories, determining group membership from unknown data. As an example, clustering techniques have been used in medical imaging systems to differentiate between tissue and blood, and in social networking analysis to group members and networks of friends or interactions (Hastie, Tibshirani & Friedman, 2013).
Dimensionality reduction	Set of new features	Unsupervised	Reduces data features to most prominent features, in an effort to improve further analysis. Dimensionality reduction is often used in facial recognition and image analysis systems (OTB Development Team, 2014, pp. 555-559).

From these problem domains, one can identify areas where machine learning may be appropriate to host enumeration. Classification of IPv6 addresses and in particular, IPv6 IIDs has been successfully carried out by Carpene, Johnstone and Woodward (IN PRESS). In this research a classification system was developed to classify IPv6 IIDs into three discrete classes; EUI-64, Privacy extensions, or Manually generated. These classes aligned with the IID generation schemes defined in de Velde, Popoviciu, Chown, Bonness and Hahn (2008). The research tested trained naive Bayesian classifiers and artificial neural network (ANN) and successfully classified over 95% of addresses. The classification system could assist in host enumeration by determining the type of a detected IID and influencing decision making about appropriate search techniques.

Machine learning processes could also aid in host enumeration by performing target address generation for search algorithms.

Common machine learning systems include the ANNs, naive Bayes classifiers (NBC), and evolutionary algorithms (EA). These systems are discussed in the coming sections.

2.3.4.1 Artificial neural networks

ANNs create a network of nodes (neurons) that correspond to the pathways in the brain (Brownlee, 2011). In a typical Feed-Forward ANN, a number of nodes are designated as input nodes, where data ingresses the neural network, and a number are designated as output nodes. The output nodes represent the sink, or egress of the data after it has traversed the neural network. The input and output nodes are then connected by a series of weighted nodes known as the hidden layer (Yu & Tsai, 2011). These weights are typically calculated randomly at first, and then optimised through the process of learning.

With supervised learning, the learning process is completed by processing a known dataset that has been correctly classified (i.e. the expected output is included with each input), and then comparing the output data to the classification. Based upon the differences between the actual output data and the expected output data, an error value is calculated. If the error value is acceptable, the system is considered to be trained and can be used to test unknown datasets. If not, then the back-propagation process is undertaken that involves retracing the steps through the neural network, and adjusting the weights at each intermediary node until the output better reflects the expected output (Brownlee, 2011). The network is recalculated and the error values reassessed until satisfactory. The process of training is important in generating a useful ANN. However, over training can also occur, and can result in potential false negatives when testing real world datasets. Unsupervised learning uses a similar process, although the training process varies since the network must discern input patterns without the assistance of pre-classified data (Brownlee, 2011).

In this research, neural networks were used in a classification system that was described in Carpene et al. (IN PRESS). Carpene et al. (IN PRESS) determined the efficacy of classifying IPv6 IID data into construction categories. The feed-forward net-

work was trained using supervised training and the back-propagation algorithm. The ANN in Carpene et al. (IN PRESS) correctly classified 96.40% of the testing dataset, although it took 29,398.108 seconds for the supervised training process to complete. Once trained, testing data took 0.375 seconds. The results of the testing indicates that the system is appropriate for classifying IPv6 IID data. The neural network classification system has subsequently been adopted and utilised for this research.

2.3.4.2 Naive Bayesian classifiers (NBC)

NBCs use probability analysis to make decisions about data. The NBC is trained using Bayes' theorem, but differs from conventional Bayes classifiers in that all identified features are treated independently (Rish, 2001). As an example, classification may reveal multiple features that comprise a result. NBCs have proven to be effective for usage in classification problems such as classifying text and diagnosing medical conditions (Rish, 2001).

Rish (2001) conducted empirical testing on NBCs to determine the reason for efficacy. Rish (2001) determines that the characteristics of the NBC allows them to be applied with reasonable success to problems of low noise (or low entropy), and situations where classification classes can be reduced to a binary choice. In the real world, NBCs have proven to be effective in classification problems, most notably in determining whether emails constitute spam (Schneider, 2003).

Carpene et al. (IN PRESS) implemented a classification system, using the multinomial and Bernoulli NBCs, to classify IPv6 IIDs into classes representing their construction types. The results of the classification was compared with that of an ANN. Our prior research conclude that whilst the ANN performed classification more accurately, the performance of the Bernoulli NBC was better, and the accuracy reasonably high (94.94% accurate with a training time of 0.391 seconds and testing time of 0.449). Although the NBC performed more efficient classification, NBCs were not utilised in this research in favour of the more accurate ANN-based classifier.

2.3.4.3 Evolutionary algorithms (EA)

Evolutionary algorithms (EA) are learning methods that are inspired by natural organic and biological processes (Yu & Tsai, 2011). A genetic algorithm (GA) is a form of evolutionary algorithm that attempts to emulate the natural evolution of species by performing mating, mutation and natural selection of data. The learning system begins by accepting an initial population of members. Each member is tested for fitness by some defined fitness function. The most fit members are then subjected to a mating function to produce an offspring. The mating function generally involves a crossover of parental attributes to produce an offspring candidate. A random mutation of the offspring is applied. The offspring forms a new generation, and the process continues until some defined conclusion is met (e.g. the number of required generations is exceeded).

EAs are often used to solve complex mathematical problems. Specifically, EAs such as GAs have been successfully applied to many search-and-sort based problems such as the one-max problem, the knapsack problem and the traveling salesman problem (Brownlee, 2011). However, EAs have not been applied to the problem of host enumeration. With off-link IPv6 host enumeration there is the potential that EAs might be suitable for the problem. The unpredictability of EAs indicate that they may be able to overcome some of the problems with host enumeration in such a vast address space. That is to say that the introduction of random mutation may allow the searched target addresses to maintain some semblance of closeness to one-another (e.g. not deviate too far from the parent) whilst also allowing unpredictability to be introduced.

2.4 Host enumeration applicability and purposes

Host enumeration serves to aid in vulnerability assessments (VAs) as a means to locate resources that can be checked for vulnerabilities. Beyond VA, host enumeration can be used to locate nodes on a network for census purposes, or as an aid in troubleshooting problems with networked devices. Additionally, host enumeration in a public scope provides an avenue for research to be conducted. IPv4 host enumeration has generated

a number of significant research efforts and contributions to knowledge. Some of these efforts are discussed in the following sections.

2.4.1 Vulnerability assessments (VAs)

VAs are exercises that agents undertake to determine the vulnerability profile of a system. In this context, a system could be, amongst other things, an application (such as a web-based application), a single server (a mail server, web server or database server), or an entire network.

Braunton (2005) defines four major components to vulnerability assessments:

1. Baseline: determine and document the normal operations of the system. The baseline must include the network baseline, including a full enumeration of network devices, services, bandwidth, usage times, etc. Of the four VA components this component is the most relevant to the research. The baseline phase defines the host enumeration exercise that is the subject of this research;

2. Audit and assess: This phase involves performing an audit of the known assets gathered from the baselining phase. The audit and assess phase has the assessor detect and document discrepancies from the baseline expectations. Various tools are available to assist with detecting, identifying and even exploiting vulnerabilities in systems, some of which are included in Table 2.4;

3. Secure the environment: The secure the environment phase sees the assessor(s) making security changes to the system in order to mitigate the detected vulnerabilities, based upon some cost/benefit and risk analysis; and

4. Evaluate and educate: The evaluate and educate phase requires the assessor to ensure security changes that are due to be applied are not further exposing the system to threats. An emphasis on the education of users and administrators is also placed. Education might include user training, lesson learned meetings, VA debriefing or other engagement activities where stakeholders can be informed about aspects of the VA proceedings (Braunton, 2005).

The first 'baseline' stage of the B.A.S.E (Braunton, 2005) VA methodology discusses the use of host enumeration in an attempt to enumerate and catalogue assets. The author places emphasis on conducting a thorough network enumeration, since rogue and uncatalogued devices present a security threat to the owner of the network (Braunton, 2005). As can be seen from the B.A.S.E VA framework, host enumeration provides a foundation for the entire assessment. A successful enumeration will provide the assessment with a more complete picture of the network's vulnerability profile.

2.4.2 Internet research

Although research surrounding IPv6 host enumeration efforts are not apparent in the literature, IPv4 host enumeration has been used for a variety of research efforts. Specifically, recently IPv4 host enumeration has been conducted in the wake of the release of major software vulnerabilities in common computer programs. The Heartbleed bug that affected the secure sockets layer (SSL) library OpenSSL in April, 2014, was subject to widespread scrutiny from the Internet community. Once the vulnerability was discovered Internet-wide scans were performed to determine which hosts were vulnerable (Durumeric et al., 2014). These findings have been published in Durumeric et al. (2014) and reveal that approximately 55% of the Alexa top one million websites (Alexa Internet, Inc., 2014) were vulnerable to exploitation using malformed OpenSSL heartbeats. The author explains that the method they used to search for the Heartbleed vulnerability was to modify the `ZMap` program to send the vulnerable heartbeat probe to target devices, and then record the response (Durumeric et al., 2014). The underlying search technique was based upon a combination of `ZMap`'s randomising function (see Section 2.2.2.4) which was used to query approximately one percent of the globally routable public IPv4 address space, as well as the Alexa top one million hosts (Durumeric et al., 2014).

Similarly, Heninger, Durumeric, Wustrow and Halderman (2012) conducted research where they performed an Internet-wide VA of weak RSA and DSA public keys. Heninger et al. (2012) study assessed the entropy of RSA and DSA public keys, to determine if the ostensibly random prime numbers used as seeds to the key generation process could be factored. To conduct this research they enumerated live hosts on

Table 2.4: A taxonomy of host enumeration and vulnerability scanning applications. Vulnerability scanners are listed along with their purpose and the algorithm they implement.

Application name	Purpose	IPv4 Support	IPv6 Support
`nmap` (G. F. Lyon, 2009)	Network scanning utility. Conducts host detection, and analyses service fingerprints to identify active services and determine OS type.	✓	✓
`masscan` (Graham, 2013c, 2013b)	Host enumeration utility. Provides high-speed enumeration of the entire IPv4 address space or parts thereof.	✓	
`ZMap` (Durumeric, Wustrow & Halderman, 2013a, 2013b)	Host enumeration utility. Provides high-speed enumeration of the entire IPv4 address space or a random sample of the address space.	✓	
`Nessus` (Beale, Deraison, Meer, Temmingh & Walt, 2008)	Vulnerability scanner. Nessus performs host enumeration and vulnerability enumeration against network devices.	✓	✓
`Nikto` (Sullo & Lodge, 2015)	Vulnerability scanner. This application will search for vulnerabilities in web servers.	✓	
`OpenVAS` (Wagner, Wiegand, Brown & Mauthe, 2009)	Vulnerability scanner. OpenVAS scans hosts for vulnerabilities. The toolkit features a network enumeration component.	✓	✓

the Internet, which they extracted the public keys from. The authors used TCP SYN probing to determine networked devices that were communicating on either port 22 (SSH) or 443 (HTTPS). Heninger et al. (2012) used a very simple `nmap` search to check the hosts. The nature of the search algorithm used for the enumeration is not discussed in the text, so it is possible either the sequential search or one of `nmap`'s randomisation functions were used during the research.

Host enumeration has also been the subject of research efforts to conduct censuses of, or enumerate networked devices connected to the Internet. Previously, Heidemann et al. (2008) conducted a complete census of the available public IPv4 address space in 2008, and was able to identify approximately 103 million live hosts. The methodology used to scan the address space involved probing all possible addresses in a pseudorandom fashion using ICMP echo probes. Similar research was conducted in 2012 with the anonymous Internet Census ('Internet Census 2012', 2013).

In this census the authors identified 450 million live nodes on the IPv4 Internet. The census was conducted using `nmap` and its scripting engine, and illegally exploited vulnerable nodes on the Internet in order to continue the census. The compromised systems were configured to execute a small binary program that helped contribute to the scanning efforts. Although unpublished officially, the results and outcomes of this research highlighted vulnerabilities in embedded devices operating on public networks. Unfortunately, the ethical aspects and validity of the research are highly questionable, even excusing the illegalities of the wide scale unauthorised computer use. Setting those social, ethical and legal issues aside, the host enumeration strategy used in this research involved using a distributed, exhaustive linear search across the entire IPv4 address space. The search was then repeated periodically to gain longitudinal data about the target systems. The delivery method of probing used a combined approach of correlating SYN scans on various common ports, such as 25, 80, 443, and sending ICMP echo requests, and ICMP timestamp requests ('Internet Census 2012', 2013).

The distributed search strategy used in 'Internet Census 2012' (2013) could be applied to widespread IPv6 host discovery. The strategy may not be practical when enumerating a single subnetwork, since there might be a bottleneck at the ingress point to that network. A bottleneck might get saturated under probing load from

distributed hosts. In any case, for the purpose of this research the assumption has been made that the receiving target in the search attempt is connected to a gigabit WAN connection. This assumption renders distributed search techniques to be impractical, since it is possible for a single node to send and receive at that rate. Distributed search techniques would be important in situations where multiple networks are targets for search efforts, such as when searching the wider IPv6 Internet. In such a situation, the combined bandwidth usage of the source nodes could be distributed over the target networks, reducing the probability of saturating target connections. This thesis aims to provide host enumeration strategies that can enable similar research to be conducted against IPv6 networks.

2.4.3 Network device identification

Wei-hua, Wei-hua and Jun (2003) discussed using ICMP for host enumeration in the context of IPv4. The paper's focus is on analysing ICMP echo replies for OS fingerprinting purposes. The research presented in Wei-hua et al. (2003) varies from the research conducted in this study, since this study is not concerned with the probing mechanism, nor the information that can be inferred from successful probes beyond an indication that a node is live on a network. Interestingly, Wei-hua et al. (2003) made a number of assertions about host operating systems and their implementation of the ICMP protocol. In particular, the authors stated that if the TTL on a packet is set to 255, then that indicates the packet originated from ULTRIX and OpenVMS OSs, whilst if the TTL is set to 128, then the OSs are Windows 95, 98, 98SE, or NT. However, the paper does not provide any indication of where these conclusions were derived from. The ICMP protocol isn't the only means by which to enumerate and identify devices. With IPv6 the IIDs assigned to a host, can form permanent or semi-permanent unique identifiers.

With the introduction of Privacy Extensions for SLAAC in RFC-4941 (Narten et al., 2007), networked nodes are able to automatically assume new IPv6 addresses at the commencement of a network session or after a specified time interval. These randomly generated addresses help to preserve the privacy of the networked device, since they're disposable, dynamically generated using stochastic methods, and only

temporarily active. Random address generation poses an administrative problem since administrators may want to catalogue unique devices on their networks.

Related work by Polcák (2014) explored the problem of trying to identify all potential IPv6 addresses on a network and correlate them to unique nodes and ultimately, the user(s) of the devices. Since a single host device participating in IPv6 communications can assume many unique IPv6 addresses, identifying unique hosts is a management problem that is difficult to overcome. The author postulates that a single method is unsuited to the task, since there are a variety of attributes that can be correlated in order to draw a conclusion about an IPv6 address' host identity (Polcák, 2014). Polcák (2014) proposes a number of approaches to identifying nodes on a network, including:

- Link-ability of identities: devices can be identified by credentials extracted from RADIUS associated with an IPv6 address to MAC address, DHCPv6, NAT or SLAAC records to link a user account to an IPv6 address;
- Local monitoring: Type 1 identity management systems (IMSt1s) are recommended for tracking user identities on a local network; Polcák (2014) described a plugin framework for IMSt1s that would enable extraction of identity information upon request from a client. Local monitoring would also involve tracking local network traffic, such as ICMPv6 requests, in order to correlate data more conclusively; and
- Remote monitoring: For remote users (e.g. VPN users, remote access users, etc.) Polcák (2014) recommends firstly extracting the MAC address from the IPv6 address (if possible). Analysing clock skew, and HTTP headers can also help to fingerprint devices (Polcák, 2014).

Polcák (2014) differs from our research since the research conducted in this study is concerned with enumerating all active hosts on a network. Unlike Polcák (2014), this research considers each unique IPv6 address to be a separate host. This assertion is made due to it being impossible to state categorically, from an outside observer's perspective, whether an IPv6 address belongs to a single host or multiple hosts. From a different vantage point (such as having assumed control of the underlying network

infrastructure), this information can be gleaned by recording a mapping of MAC address and unique IPv6 address pairs, or through the other methods described in Polcák (2014). However, such information is not available to off-link actors.

2.5 Conclusion

From the review of the available literature it has been determined that off-link IPv6 host enumeration has been scarcely researched. The available literature pertaining to published host enumeration exercises relate, almost exclusively to IPv4 host enumeration. The apparent consensus is that off-link host enumeration against IPv6 networks cannot be performed successfully due to the large address space.

There is a wealth of prior research that focuses on developing and implementing complete host enumeration strategies for IPv4, and to a lesser degree, IPv6. Very little of the prior research focuses specifically on the search algorithms being employed.

There are validated approaches to enumerating devices on off-link IPv4 networks, including through linear searching and randomised searching of the target address space. These approaches have not been validated for usage against IPv6 networks. This research aims to remedy this by testing the approaches against IPv6 networks.

Machine learning as an approach to host enumeration, either against IPv4 or IPv6 networks, has thus far not been applied. Machine learning has proven to be effective in optimisation problems, such as the travelling salesman problem, classification problems, and in situations where there are known-unknowns such as with clustering problems. These attributes imply that the approaches could be used to assist with host enumeration search operations.

THIS PAGE HAS BEEN INTENTIONALLY LEFT BLANK

Chapter 3

Research Methodology and Design

3.1 Appropriateness of the design

In order to validate the appropriateness of the research design and methodology chosen, it is necessary to understand the world views that underpin scientific research. Creswell (2009) identifies five main philosophical paradigms that influence the way research is designed and conducted; positivism and post-positivism, constructivism, transformative, and pragmatism. Table 3.1 illustrates a modified taxonomy of paradigms from Creswell (2009) that summarises the relationship between these five philosophical perspectives.

Associated with each of these paradigms is the ontological perspective, which explores the nature of truth, existence and reality, as well as the epistemological perspective which is concerned with how knowledge is actually created. Table 3.2 provides a summary of the relationships between the aforementioned research paradigms, truth, knowledge and general modes of inquiry.

Empirical research suggests that knowledge can be created from observations of phenomena (Graziano & Raulin, 2004; Jackson, 2009). Although all empirical research holds this foundational belief, there are subtle differences between how the positivist and post-positivist research paradigms interpret this understanding. Positivists

Table 3.1: Table of the five major world views that influence contemporary research design (Creswell, 2009, pp. 6).

Research paradigm (world view)	Major elements of world view
Positivism	• Determination • Reductionism • Empirical observation and measurement • Theory verification
Constructivism	• Understanding • Multiple participant meanings • Social and historical construction • Theory generation
Transformative	• Political • Empowerment issue orientated • Collaborative • Change-orientated
Pragmatism	• Consequence of actions • Problem-centered • Pluralistic • Real-world practice orientated

believe that it is possible to create objective knowledge through carefully controlled experimentation. This belief is contrary to the post-positivistic perspective which acknowledges that, while the aim is to gain objective knowledge, the results may not be free from potential bias even with careful control. However, both positivist and post-positivist paradigms hold that there is a distinct relationship between cause and effect (Creswell, 2009). As a result the modes of inquiry used in such research rely on comparing research subjects exposed to varying conditions (Graziano & Raulin, 2004). Typically, empirical research involves the use of experimental processes to invoke and observe phenomena, however survey techniques can also be applied.

Table 3.2: Research paradigms and their associated ontologies, epistemologies, research methodologies, and modes of inquiry. Adapted from Creswell (2009), Fien (2002), Mackenzie and Knipe (2006), Donna M Mertens (2010), Potter (2006).

Research paradigm	Ontology	Epistemology	Research methodology	Modes of inquiry
Positivism	Anything that cannot be observed cannot be true.	Assumes that knowledge is simply waiting to be discovered	Quantitative	• Experimental • Quasi-experimental • Surveys and tests
Post-positivism	Absolute truth can never be found therefore, failing to reject a hypothesis is the main objective.	Knowledge is acquired through the observation of phenomena.	Quantitative	• Experimental • Quasi-experimental • Surveys and tests
Constructivism	Reality is based upon human interpretation. It is not limited to an individual, instead is constructed socially.	Knowledge is subjective and generated through the interaction of the researcher and their subjects	Primarily qualitative, can also be quantitative.	• Interviews • Observations • Document reviews • Visual data analysis
Transformative	Recognises influence of privilege when determining reality. "Multiple realities are shaped by social, political, cultural, economic, ethnic, gender, disability and other values" (Donna M Mertens, 2010).	Knowledge is subjective. Power and privilege has a heavy influence on knowledge.	Mixed methods	• Variety of tools (e.g. surveys, case studies, and document reviews). Particular attention given to avoiding social issues when conducting research.
Pragmatism	Truth is established based upon what works at the time. It is not limited to the experiences or interpretations of the individual	Knowledge is generated from groups of individuals engaging with their environments.	Mixed methods	• May include modes of inquiry used in the positivist or constructivist paradigms. E.g. interviews, observations and testing, and experiments.

Social constructivists hold the opinion that knowledge is dependent on individual and social human experiences. The underlying epistemology of constructivism is that knowledge is subjective, and that the relationship between the researcher and the object of enquiry is what drives the generation of knowledge. Constructivist studies generally involves human participants as the subjects of the research. As a consequence of this belief framework, constructivist research is well suited to qualitative studies. Research conducted may make usage of a number of modes of inquiry in order to address a particular research problem. A study may employ surveys, interviews, document review, observations, or visual data analysis as a means to address the research problem.

The transformative paradigm is a world view that has a social justice focus (Donna M Mertens, 2010). The paradigm arose in response to deficiencies in social constructivism's ability to address problems of a social or political nature. The paradigm provides a framework for research undertakings that address problems of inequality and injustice within societies (Donna M. Mertens, 2007). With the dynamic nature of research problems that transformative research is posed with, a mixed methods approach is appropriate.

A pragmatic world view assumes no single methodology or mode of inquiry is appropriate in generating all knowledge. A pragmatic approach views research as independent undertakings that should consider and utilise any approaches that best address the primary research problem. Consequently, pragmatic research generally employs a mixed-methods approach, combining the strengths of both qualitative and quantitative methodologies wherever necessary.

This research was carried out from a post-positivist perspective for the development of knowledge. It was assumed that knowledge can be inferred and acquired through the process of experimentation and observation. Coming from a post-positivistic philosophical perspective, it was appropriate to view knowledge generation empirically. With this epistemological backing, a quantitative research methodology was chosen to address the research goals. The research questions could not be appropriately answered using qualitative means, therefore the research was conducted using quantitative methods.

Experimental methods were chosen to address the research goals. Experimental

research involves constructing and performing experiments that invoke the occurrence of phenomena. Changes in variables are recorded in order to produce results. From these results, inferences and conclusions can be drawn about the nature with which the experimental variables interact with each other. Experimental research was chosen for the study because it was the most suitable way to address the primary research questions of the study. In particular, this research involved applying algorithms to solve the problem of IPv6 address space enumeration and then testing new methods against existing methods. This distinction of measuring numerical changes necessitates the usage of quantitative research techniques.

This research began with the following research questions:

RQ1 *Can networking devices be enumerated on 64 bit IPv6 subnetworks using host discovery techniques?*

RQ2 *Are stochastic searching methods more efficient than deterministic searching methods when enumerating IPv6 hosts within a single 64 bit subnetwork?*

RQ3 *Do stochastic address allocation schemes within a single 64 bit subnetwork inhibit IPv6 host enumeration strategies?*

RQ4 *Can machine learning search methods be used to enumerate devices on a 64 bit IPv6 subnetwork?*

RQ5 *Are machine learning searching methods more efficient than non-machine learning based methods when enumerating IPv6 hosts within a single 64 bit subnetworks?*

After considering the literature surrounding the research topic, the following hypotheses were then formulated. These hypotheses aim to provide answers to the research questions posed:

H1 *"Search techniques are unable to enumerate networked devices on 64 bit IPv6 subnetworks."* (answers RQ1)

H2 *"Methods that employ random sampling do not perform better than methods that do not employ random sampling for IPv6 host enumeration."* (answers RQ2)

H3 *"Randomly generated interface identifiers do not affect the performance of IPv6 host enumeration search algorithms."* (answers RQ3)

H4 *"Search methods that employ machine learning techniques cannot be used to enumerate devices on 64 bit IPv6 subnetworks."* (answers RQ4)

H5 *"Search methods that employ machine learning do not perform better than search methods that do not employ machine learning for IPv6 host enumeration."* (answers RQ5)

In order to test the hypotheses and address the primary research questions, experiments needed to be developed and conducted. During the development process, three subcategories of experimental methods were considered: natural experiments, field experiments, and laboratory experiments. Natural experiments require performing experiments in a natural setting, unadulterated by the researcher. This form of research inquiry was not appropriate for the study, since it would require testing against live IPv6 networks not governed by the researcher, and where the specific configurations are largely inferred or unknown. Field experiments are similar to natural experiments in that they require the use of live, yet artificial, situations where not all variables are under the control of the researcher. Field experiments were considered for this project, and preliminary research involved using this mode of inquiry. However, it was determined that since the experiments undertaken in this research required probing networked nodes, performing the experiments against live computer systems over a live network would incur significant and unnecessary time penalties. Additionally, since the independent variables were the algorithms that were being applied to the problem, live testing would have introduced uncontrollable independent variables. This would have impacted on the repeatability and validity of the results, without providing any benefits to the testing of the enumeration algorithms.

Laboratory experimentation with computer simulations, therefore, was the most appropriate option for experimenting. A laboratory setting allowed the author to control more aspects of the experimentation process, which increased the internal validity of the study (Williamson & Johanson, 2013). All experiments conducted throughout the research were laboratory experiments, where the object of inquiry was the algorithm

being tested. Following the research process recommended by Mackenzie and Knipe (2006) the research project was designed (Figure 3-1).

3.2 Design and procedure

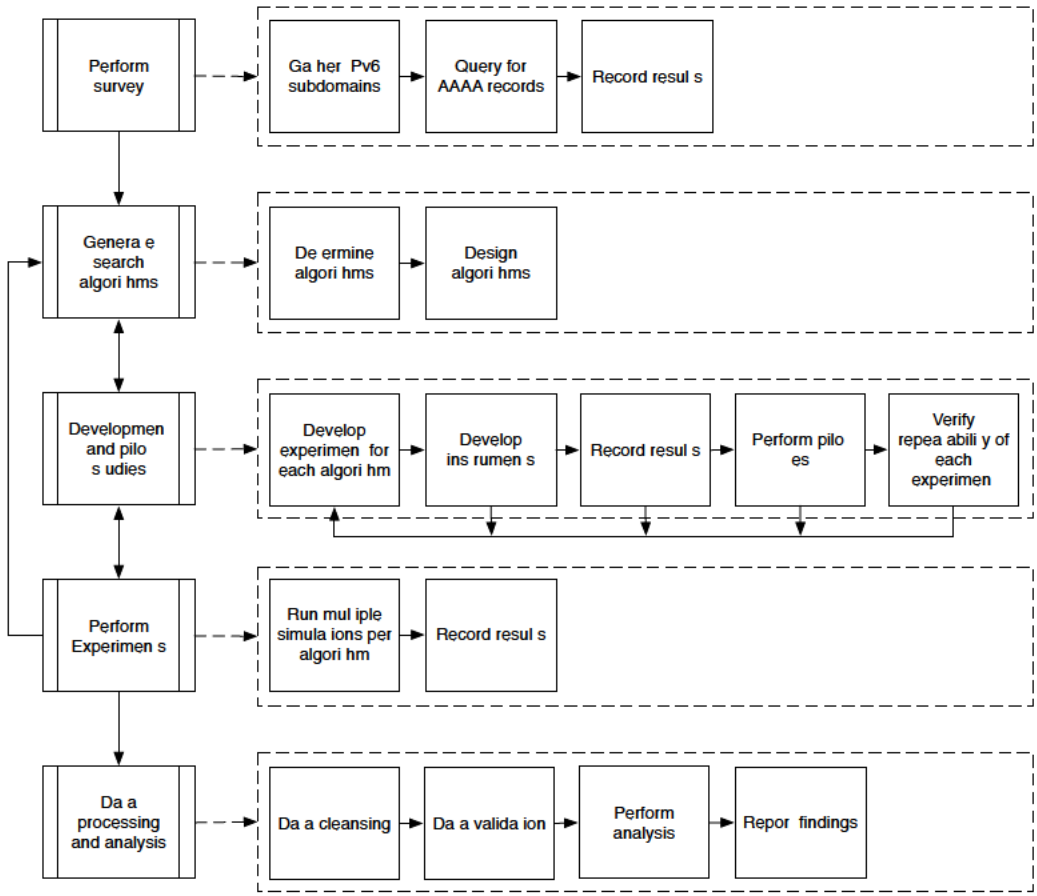


Figure 3-1: The research framework followed during the investigation outlining the five main phases, and the associated subprocesses of the research project.

A framework (depicted in Figure 3-1) was developed from Mackenzie and Knipe (2006) that outlined five distinct phases of the research process. The five phases described in the framework include:

1. The survey phase, where a survey was conducted to determine the ways IPv6 is used in the real-world situations.

2. The generation of search algorithms, which involved designing the algorithms that would form the subjects of the experimentation.

3. The developing of experiments and undertaking of pilot studies, which was concerned with realising the designed algorithms into testable experiments and subsequently trialling them.

4. The experimentation phase, where the experiments were conducted and the results recorded.

5. The analysis of results phase, where the data gathered in the experiments were transformed and inspected until information was gathered.

3.2.1 Research variables

With the goal of assessing the efficacy of various IPv6 host enumeration search strategies in mind, the variables for the experiments were set. The common variables that underlined all of the experiments performed in this study are described in this section.

3.2.1.1 Dependent variables

In an effort to test the research hypotheses and answer the research questions, relevant data were gathered from the experimentation. To this end, the following variables were measured in each experiment during the study:

- The time, in seconds, that each search exercise took to complete.
- The number of successful probes that each simulation yielded.

In addition to the primary dependent variables above, the following variables were measured and recorded during the experiments:

- Each valid target IPv6 address that was probed.
- The total number of probes transmitted in each simulation.

3.2.1.2 Independent variables

The independent variables for the overarching research project were the algorithms that were applied to the problem. These algorithms were designed to test the hypotheses of the study in order to answer the research questions:

- Linear search algorithm.
- Stripe search algorithm.
- Monte Carlo search algorithm.
- Pattern-based search algorithm.
- Genetic algorithm.
- Adaptive search algorithm.

3.2.1.3 Controlled variables

In order to maintain repeatability and isolate the changes in dependent variables, a number of experimental variables were controlled. These controlled variables are as followed:

- The surveyed list of valid IPv6 addresses that algorithms were tested against.
- The pseudo-randomly generated list of valid IPv6 addresses that the algorithms were tested against.
- The computing equipment that were used to conduct the experiments.
- The software packages and software libraries that aided in developing instrumentation and performing the experiments (see Section 3.2.4.3).
- The search size for all algorithms (address 0 to address 2^{64} exclusive).
- The maximum number of probes each search exercise could transmit. For the purpose of this research, a ceiling on the maximum number of probes was set at 2^{32} so as to conduct the search in an acceptable time frame. This reduced the search space for each algorithm from 2^{64} nodes down to 2^{32} nodes. To put this in perspective, at a conservative rate of 1,000,000 probes per second, searching a 2^{32} address space would take approximately 71.6 minutes to complete. Performing the same exhaustive search across a 2^{64} bit address space would take 584,942.42 years. Note: for pilot studies the maximum number of probes each simulation could transmit was limited to 10,000 probes.

- The delivery of probes. In real-world scenarios a number of methods may be used to probe a host and determine if it is alive. Techniques such as TCP connect scans, TCP SYN scans (half-open scanning) or ICMP Echo probing, amongst others may be applied to such situations. This study chose to use a local emulation of probing a host. The probing was conducted by performing a membership test to see if the target host address is contained within a list of known valid host addresses. If the host was valid (i.e. it was a member of the set of valid hosts) it would be added to an array of detected nodes.
- The number of simulations performed for each experiment.

3.2.1.4 Compound variables

Many of the experiments conducted made use of pseudo-random number generators (PRNGs) to provide random influences where required. There are many variables that can influence the results when experimenting using stochastic methods. In this study the primary compound variables pertain to the stochastic and random elements of each algorithm. To preserve repeatability and reproducibility, important aspects of the PRNG were influenced and recorded at the commencement and conclusion of each experiment. Each Python program's PRNG was seeded with a randomly selected value obtained via the operating system's random number source (i.e. the `/dev/urandom` device on Linux and Unix systems). In the case of programs written using C the PRNG was seeded using four bytes from the `arc4random` function. The starting states of the PRNGs were recorded immediately after seeding. Additionally, the final states of all PRNGs were recorded at the conclusion of the experiment. With this information it was possible to accurately recreate the experiments and validate the results.

3.2.2 Phase 1: Perform survey of IPv6 usage

In the first phase of the research a survey was conducted of IPv6 usage data in real world situations. The intention of the survey was to assess real world usage habits of the protocol, as well as to provide valid samples to be used as the experimentation data.

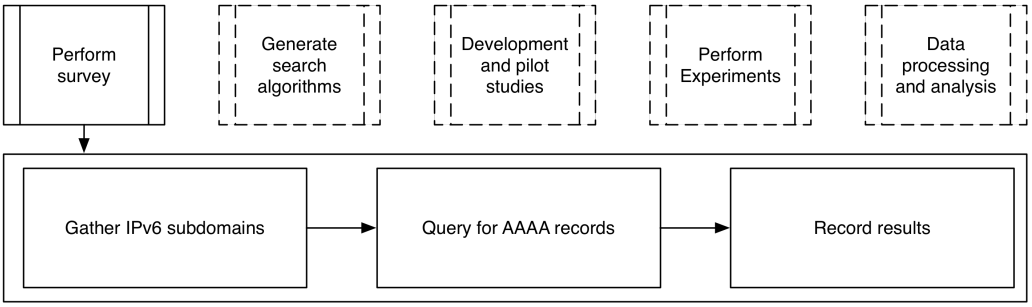


Figure 3-2: A breakdown of Phase 1: the survey phase of the research highlighting the subprocesses involved. Phase 1 consisted of three major subprocesses; the gathering of IPv6 subdomains; querying the subdomains for IPv6 AAAA host records in public DNS ; and recording the results from the DNS enumeration.

The survey was conducted over a period of 454 days. Data were collected through the use of a passive enumeration of DNS. The DNS enumeration involved querying public DNS servers for IPv6 AAAA host records. The requirements for the survey was that a large number of potential subdomains were queried for AAAA records. These subdomains were obtained from a number of sources (see Table 4.1 on page 111) over a total of 969 intervals. The procedure used when conducting the survey is detailed below.

1. Gather list of potential subdomains: the list of subdomains used for the DNS enumeration attempt were first gathered from the lists of subdomains listed in Table 4.1. Additionally, for each unique root domain name gathered, the following potential subdomains were also compiled into the list: `ipv6.<domain>`, `www.<domain>`, `mail.<domain>`, `intranet.<domain>` and `vpn.<domain>`. These prefixes were chosen since they are commonly used for public Internet services.

2. Each subdomain from the list was then queried for an AAAA IPv6 host record using public DNS servers.

3. The results were gathered into a database of IPv6 addresses, along with the subdomain the address corresponded to, a timestamp and the source list of domains the result originated from.

In addition to the surveyed IPv6 dataset, a randomly generated dataset was created to test the efficacy of the algorithms against unpredictable data. The random dataset was constructed by using a PRNG to produce random, non-unique, integer values,

between 0 and 2^{64}, 50,000 times. From there a random number of IP addresses were chosen for duplication. Duplication was performed to ensure that a subset of addresses were not unique so weighted choices could be influenced by frequency of occurrence. A random number of addresses between 0 and 1,000 were chosen for duplication. Each selected address was then duplicated a random number of times between 0 and 1,000.

These datasets were used as the candidate target IPv6 networks for experimentation. The surveyed dataset represented a network with node addresses allocated in ways representative of real world networks. The randomised dataset represented a network where IPv6 addresses were assigned by taking advantage of high entropy allocation schemes, such as SLAAC with privacy extensions, or CGAs.

3.2.3 Phase 2: Generate search algorithms

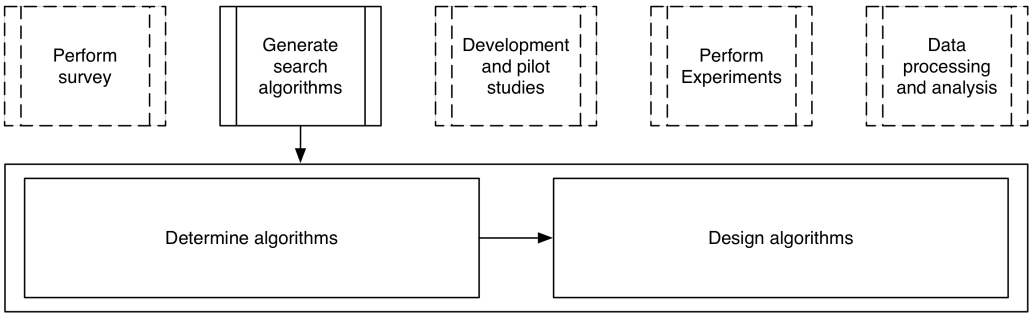


Figure 3-3: A breakdown of the the Phase 2: Generate search algorithms phase of research phase including major processes involved. Phase 2 of the research involved designing and determining algorithms that would be suitable for searching IPv6 networks. These algorithms were subsequently used within the research.

Based upon the results of the survey data, appropriate algorithms were designed that formed the independent variables that would be tested. Each algorithm was the subject of one or more experiments. Each experiment involved testing an algorithm against target datasets of valid IPv6 addresses and recording the results. The algorithms that were chosen to be tested during the study are now described.

3.2.3.1 Linear search algorithm

The linear search algorithm designed for the study was based upon a standard sequential search, although unlike an exhaustive search, the algorithm used in this study only searches a limited address space. A diagrammatic representation of an example linear

search across a hypothetical address space is presented in Figure 3-10(a).

Since the algorithm searches a reduced scope of addresses, the algorithm, which is depicted in Figure 3-4, requires a starting point to begin the search. The process of the linear search algorithm designed for this research is as follows:

1. Select a start and end point such that $0 <= start\_point <= (2^{64} - max\_probes)$ where max_probes is an integer representing the maximum number of probes to deliver (in this case 2^{32}). The end point was realised such that $end\_point = start\_point + max\_probes$

2. Probe each discrete address from $start\_point$ to end_point

3. Return results

The linear search algorithm can be initialised in three distinct ways; by starting from address 0 in the address space; starting from a randomly chosen address in the address space; or by starting from a point in the address space selected using a weighted random choice, based upon existing weights for the entire address space (determined in Section 3.2.5).

3.2.3.2 Stripe search algorithm

The stripe search algorithm is a modified linear search that chunks the search space into evenly distributed searchable and unsearchable regions, an example of which is portrayed in Figure 3-10(b). From there, a simple linear search is performed on each chunk. The algorithm that was designed for the research is reflected in Figure 3-5. This algorithm is loosely based upon the comb sort algorithm, which is in itself a slight variation of the bubblesort algorithm (Lacey & Box, n.d.). Although a typical comb sort operation would involve multiple passes over a single dataset (combing different chunks over each pass), when translated to a search operation over the IPv6 address space it was efficient and prudent to perform the comb a single time.

The algorithm also derives influence from the reverse IP-sequential approach described in Leonard and Loguinov (2010) and discussed in Section 2.2.

The process of the stripe search algorithm is as follows:

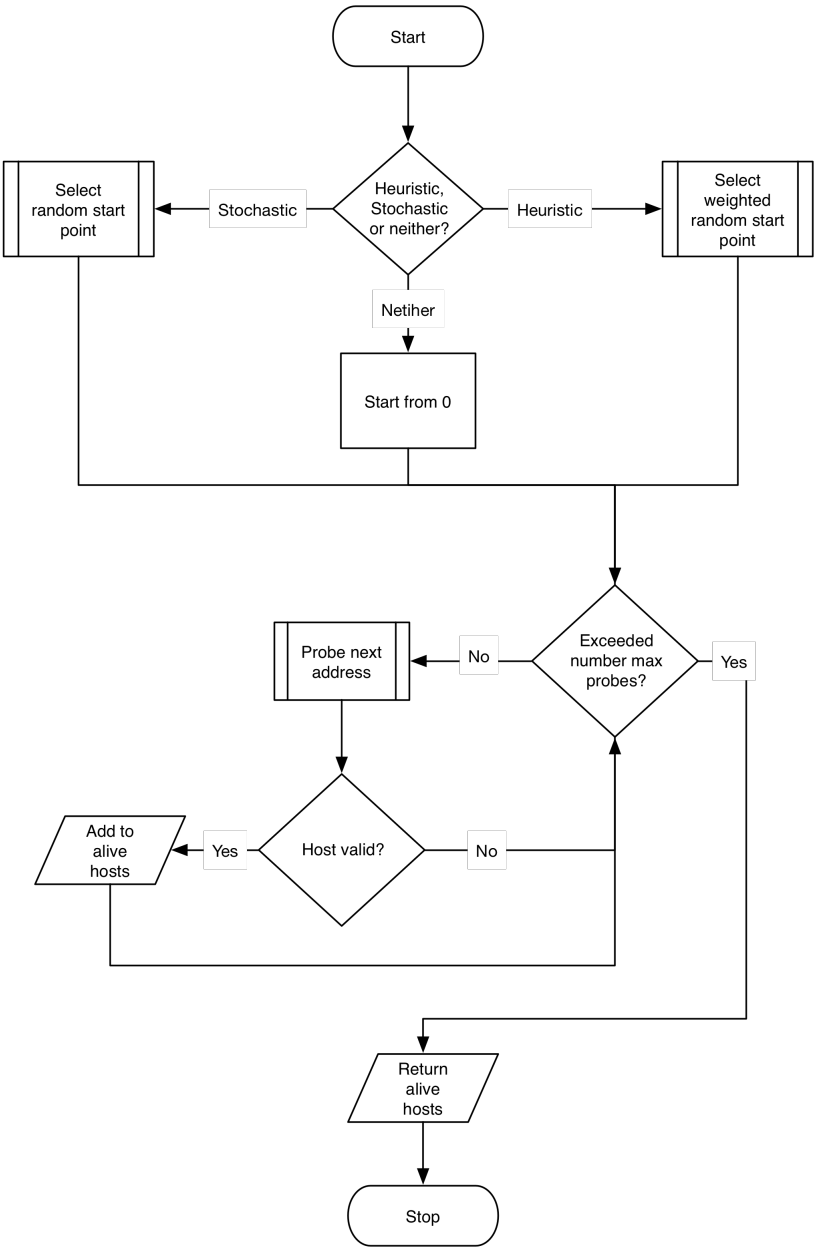


Figure 3-4: Flow chart depicting the major processes involved within the linear search algorithm that was designed for this research.

1. Select a start and end point such that $0 <= start\_point <= (2^{bin\_size} - n\_probes)$ where bin_size is an integer $0 <= i <= 64$ and n_probes is an integer representing the number of probes to send per interval (bin) such that $0 < i <= bin\_size$. The end point was realised such that $end\_point = start\_point + n\_probes$

2. Probe each discrete address from $start\_point$ to end_point

3. Update start and end points such that $start\_point = start\_point + 2^{bin\_size}$ and $end\_point = end\_point + 2^{bin\_size}$.

4. Repeat from step 2 until all bins have been combed.

5. Return results

3.2.3.3 Monte Carlo search algorithm

The Monte Carlo search algorithm (featured in Figure 3-6) was designed to perform an effectively random search of the entire 64 bit address space. The Monte Carlo algorithm was designed in an attempt to test Hypothesis *H2* by randomly sampling addresses from the address space. The random sampling technique is similar to that which has been applied to IPv4 host enumeration by applications such as `masscan` or `zmap` described in Section 2.2.2. A diagrammatic representation of a potential spread of probes is presented in Figure 3-10(c).

The algorithm designed for this research did not permute the sequential address space (as is the case with the Generalised Feistel algorithm posed in Graham (2013c)) instead the targets were chosen by pseudorandom choice to reduce computational overheads. As a result it was required that the algorithm maintain a minimal state table of valid addresses that had been checked.

The algorithm can be broken into four major steps.

1. Pick a pseudorandom integer between 0 and 2^{64}.

2. Probe address

3. Repeat from step 1 until maximum probes have been sent.

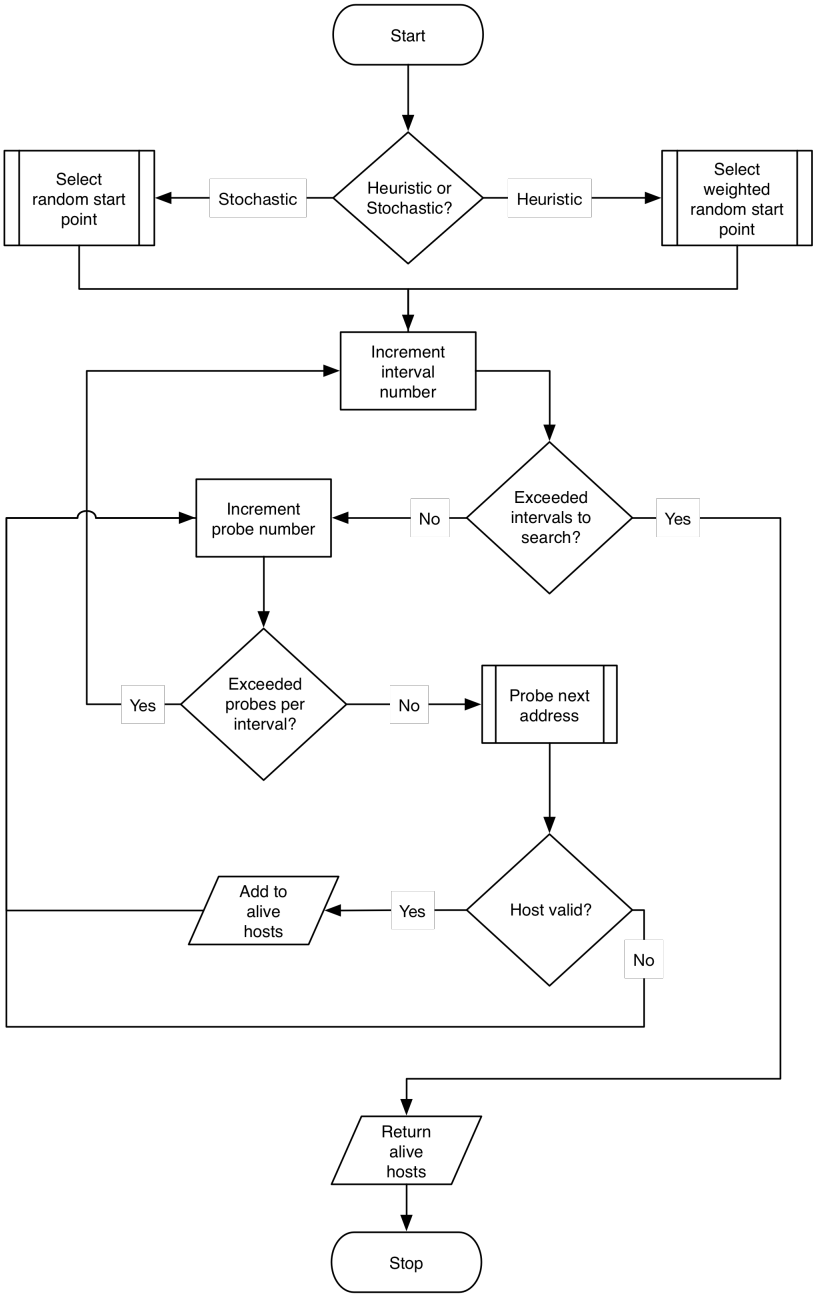


Figure 3-5: Flow chart depicting the major processes involved within the stripe search algorithm that was designed for this research.

4. Return the results

In experiment 1, random sampling was carried out as per the progression of the PRNG, without influence. In experiment 2 the random sampling was influenced by weightings that were applied to the address space. Based upon the provided weights, the random selection would be more likely to select addresses within the heavily weighted regions of the space. The process for determining the weights used is detailed in Section 3.2.5.

3.2.3.4 Genetic algorithm

The main objective of this experiment was to determine whether applying evolutionary algorithms, in particular genetic algorithms, to the problem of IPv6 host enumeration was an appropriate strategy. This algorithm was posed in an attempt to test Hypothesis *H4* and Hypothesis *H5*. Machine learning algorithms have not yet been applied to the problem of IPv6 off-link host enumeration. In doing so a genetic algorithm was generated, along with associated fitness tests, which were then applied to the dataset.

The genetic algorithm (GA) had eight main operations.

1. Generate an initial population. The method of generating the initial population varied depending on the experiment conditions. These methods were:

   E1: For Experiment 1, the organisms in the initial population were initialised to be zeroes, meaning every organism in the starting population was 0.

   E2: For Experiment 2, the initial population were randomly chosen to be an integer such that $0 <= i < 2^{64}$).

   E3: For Experiment 3, the starting population was constructed by taking integers in the search space and creating parent pairs from them. The method used to obtain parent1 ($p1$) and parent2 ($p2$) involved $p = random(0 <= i < bin\_size)$, so that $parent1 = p * 2^{bin\_size}$ and $parent2 = ((p+1) * 2^{bin\_size}) - 1$ where bin_size was an integer $0 <= i < 64$. This ensured that the parents were the start address and end address of a particular bin in the search space.

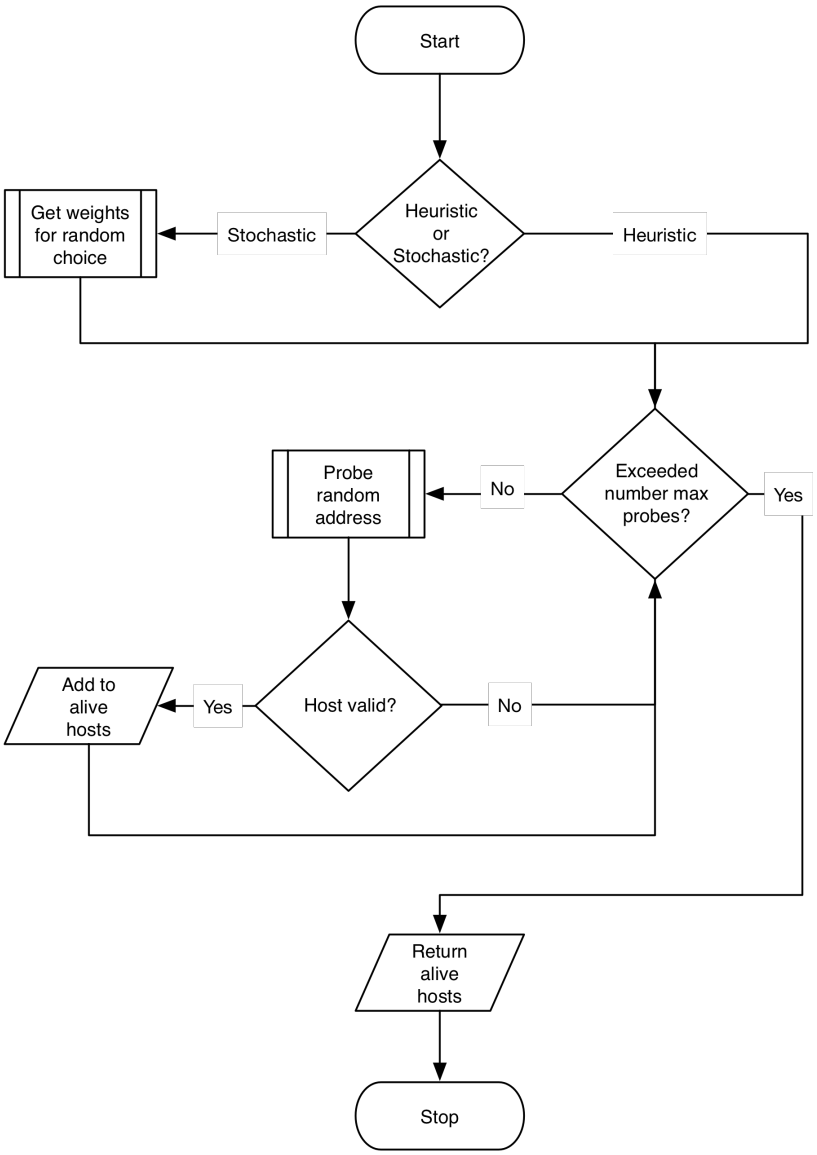


Figure 3-6: Flow chart depicting the major processes involved within the Monte Carlo search algorithm that was designed for this research.

2. Generate a model organism. The model organism represents a set of alleles that the algorithm considers perfect, and influenced how the fitness function scored an organism. The model organism was generated pseudorandomly during the initial population generation phase.

3. Probe each address in the population

4. Evaluate the address using a fitness function. A fitness function, described in Algorithm 5, was used to determine the fitness score of each organism. This fitness score influenced the likelihood that an organism would be chosen as a candidate for breeding. The fitness function first considered the outcome of the probing attempt.

    - If the probe was successful, and the organism had not been probed before, a fitness score of 64 was applied to the organism. This fitness score indicated that each allele (bit) of the organism was ideal. If the organism had been previously probed, a score of $64 * 0.9$ was applied to the organism, which indicated that most of the alleles were ideal but provoked further mutation. This helped to discourage the algorithm from falling into a local maximum in an effort to prevent the same organism from being repeatedly created.
    - If the probe failed, then each allele of the organism was compared to the corresponding allele of the simulation's model organism. Where the organism's allele matched the model organism's allele, the fitness score incremented by 1.

    The fitness function described in Algorithm 5 accepted the following four variables as input:

    - BinaryIndividual: An array of 64 binary integer values representing the binary digits of the IID;
    - ModelOrganism: An array of 64 binary integer values representing the binary digits of the model IID;
    - RealHosts: An array of unsigned 64 bit integer values representing the alive nodes on the target network; and

- ValidHits: An array of unsigned 64 bit integer values representing the networked nodes that have been successfully probed.

5. Once evaluated, candidate parents were chosen to mate. Selecting of candidate parents was conducted using a tournament-style selection process that was influenced by the fitness value of each parent. The most fit parents were chosen for breeding.

6. Next the candidate parents were mated. Mating consisted of taking two candidate parents and performing a crossover of the binary representation of the parents at a single pseudorandomly chosen interval between bit 0 and bit 63. The output of this function represented new offspring. If $0 <= mutation <= random <= 1$ a mutation function was performed against the offspring that would flip each bit from bit_0 to bit_{63} with a 2% probability per bit. This process was repeated until the required number of offspring per generation was achieved.

7. Repeat from step 3. with the new offspring population until the desired number of generations or total probes delivered has been reached.

8. Return the results.

3.2.3.5 Pattern-based algorithm

The pattern-based algorithm takes influence from Hauser (2006) and was designed to test Hypothesis *H2*. It incorporated the common methods of IID generation that are implemented by humans that configure networking equipment. This algorithm exploits this lack of entropy in address generation to enable host enumeration by using heuristics to generate targets based upon a pre-existing understanding of common IPv6 usage patterns. The following construction patterns are being targeted with the pattern-based algorithm:

- Low range Incremental IIDs: This category includes addresses that begin at low numbers in the address space (e.g. `::`) and increment upward.

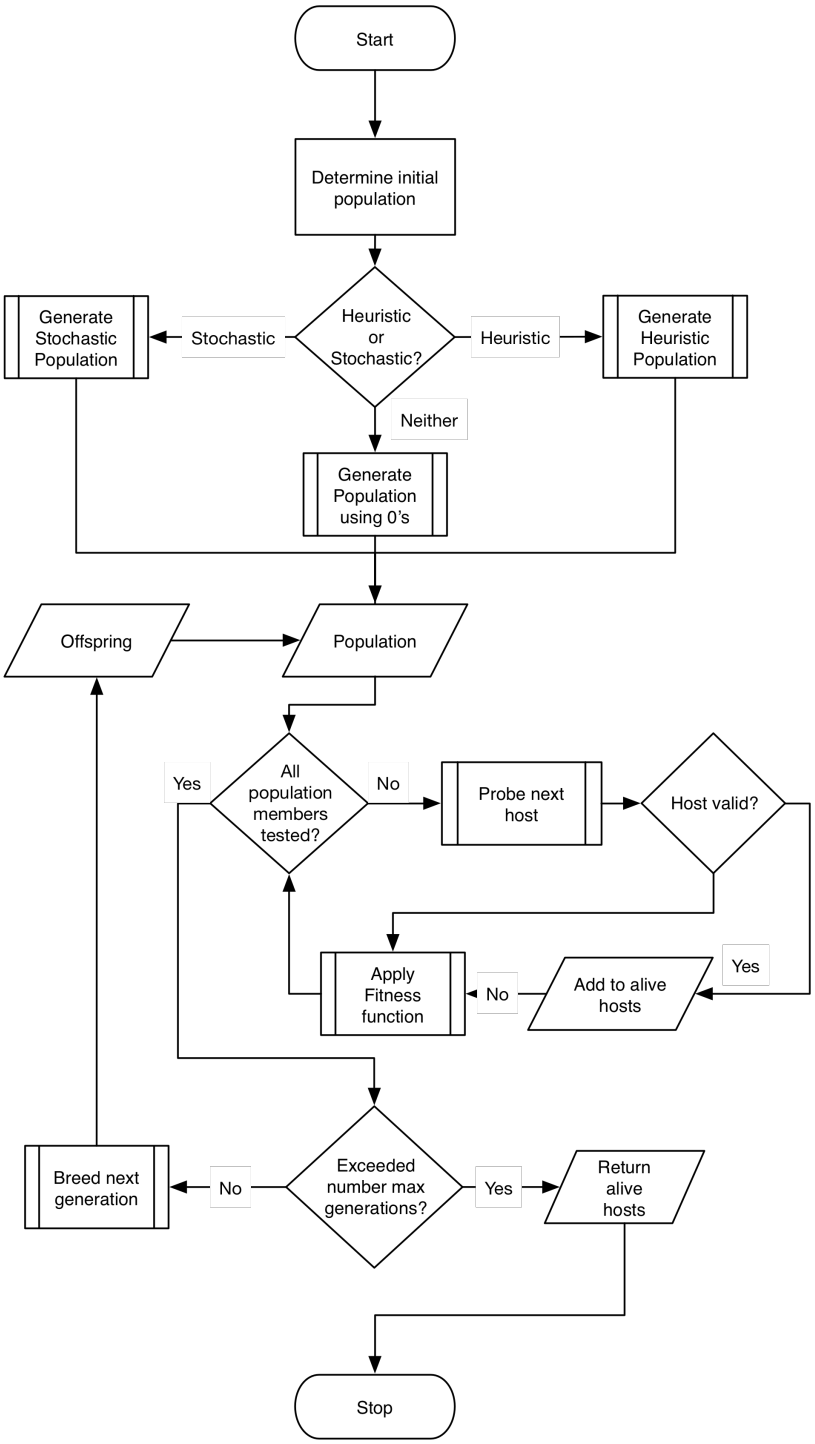


Figure 3-7: Flow chart depicting the major processes involved within the GA search algorithm that was designed for this research.

```
Data: BinaryIndividual, ModelOrganism, RealHosts, ValidHits
Result: FitnessValue
Individual ← BinaryArrayToInteger(BinaryInteger);
if Individual in RealHosts then
    if Individual in ValidHosts then
        // This node has been previously probed, return a lower
           fitness value to discourage falling into a local maximum
        FitnessValue ← 0.9 * 64;
    else
        // First time we have probed this node, and since the host is
           alive, it is assigned a fitness value of 64
        n ← CountElemsInArray(ValidHosts);
        ValidHosts[n] ← Individual;
        FitnessValue ← 64;
    end
else
    for i ← 0 to 63 do
        if BinaryIndividual[i] == ModelOrganism[i] then
            FitnessValue ← FitnessValue + 1;
        end
    end
end
return FitnessValue;
```

Algorithm 5: Pseudocode for the fitness function used within the genetic algorithm to evaluate and score target host addresses. The function returns a continuous value between 0 and 1 representing the fitness score of the address for each allele

- IPv4-converted-IPv6 addresses: These addresses include the 2^{32} possible addresses that can be constructed by colon-separating an IPv4 address (for example, the 32 bit IPv4 address `192.168.1.1` could be converted into the 64 bit IID `::192:168:1:1`).

- Wordy addresses: This category includes IPv6 IIDs that have been constructed using common hexadecimal word substitutions (e.g. `::dead:beef:cafe:face`).

The pattern-based algorithm generated a target list using the methods mentioned above, and then probed each of the addresses (as depicted in Figure 3-8). The pattern-based algorithm designed for this research deviates from the one presented in Hauser (2006) and the `alive6` program by targeting different address construction methods such as the IPv4-converted-IPv6 style addresses.

3.2.3.6 Adaptive heuristic search algorithm

The adaptive heuristic search algorithm represents a significant portion of the contribution to knowledge made by this research. This algorithm (featured in Figure 3-9 on page 87) was designed to test Hypothesis *H4* and Hypothesis *H5*. The algorithm utilised machine learning to influence its decision making about which addresses to target. This approach is in contrary to existing algorithms, as it harnesses information that is known about the valid addresses it probes (such as construction classification type). Typically, host enumeration strategies, such as those discussed in Chapter 2, construct a deterministic list of targets at runtime rather than adjusting the target list based upon what address types are known-knowns. This algorithm is designed to adapt to successful and unsuccessful probes, and alter its actions accordingly. An example of a probing pattern of the Adaptive heuristic algorithm is included in Figure 3-10(d). The algorithm uses both passive and active host enumeration in order to improve the probability of successfully identifying nodes on a network.

The adaptive heuristic search algorithm's operations can be deconstructed into five steps.

1. First, a reconnaissance operation is conducted against the target network. This reconnaissance operation constitutes the passive enumeration strategy, and looks

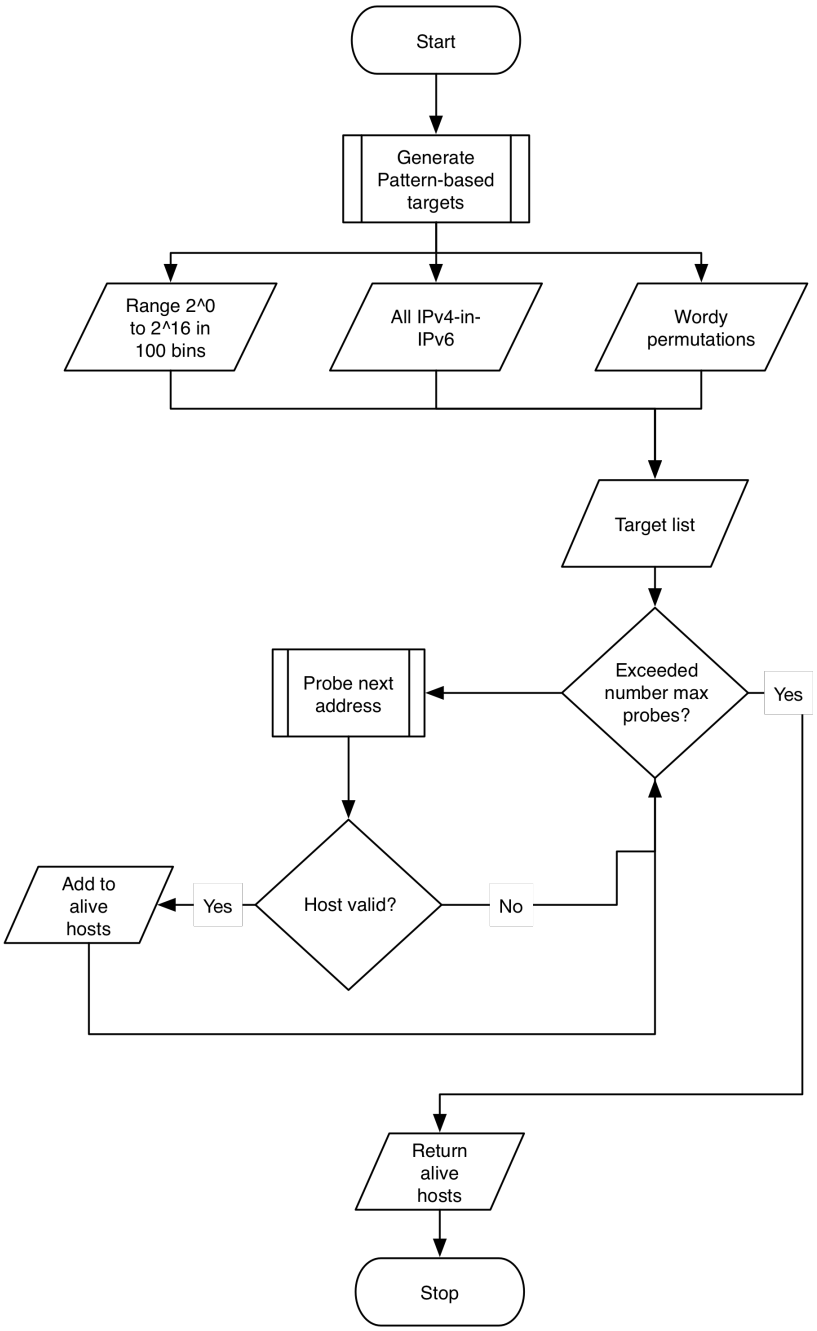


Figure 3-8: Flow chart depicting the major processes involved within the heuristic pattern-based search algorithm that was designed for this research.

to public information sources to gather any additional hosts that are readily available. The motive behind the reconnaissance is twofold, firstly it lessens the initial target address space and increases the probability of positive results. Second, the gathering of known data is used to influence the progression of the algorithm into more probable areas of the address space. In the study the author used a portion of the known dataset as the 'results' of the reconnaissance data. This dataset was attained by performing a random sampling of the input dataset of IPv6 addresses at the commencement of the experiment. The reconnaissance data differed per experiment. When performed under stochastic conditions, the reconnaissance data were acquired by random selection without weighting, whereas in the heuristic experiments the data were acquired through a weighted random choice. As described in section 3.2.5, the input data were weighted based upon the occurrence count of each unique IID. The data gathered from the reconnaissance phase was transferred into a list of targets.

2. Second, the IPv6 addresses in the target list are passed through a classification system. The classification serves to distinguish the type of interface identifier the target IPv6 address has. This was used to determine what IID construction technique was employed in an attempt to influence the proceedings of the algorithm. The classifier used in the study was based upon an ANN classification system modified from Carpene et al. (IN PRESS). This classifier takes the input address in integer format and returns a classification of either EUI-64, Incremental or Stochastic, representing the three distinct categories of address construction relevant to the research.

3. Next, the classified addresses are sent to separate processing functions based upon their classification. For EUI-64 address types, the MAC address is inferred from the modified EUI-64 constructed address as per Carpene and Woodward (2012), Groat et al. (2010), Thomson et al. (2007). A range of discrete addresses was then constructed using integers sharing a common MAC OUI. As noted in Carpene and Woodward (2012) fleets of workstations or servers in corporate environments or data centres may use network interfaces derived from a common batch of devices.

For incremental addresses, a simple range of discrete addresses about the target was added to the list of hosts to test. Finally for stochastic addresses, firstly a binary entropy test is conducted to determine if the classification is accurate, and if it passes a number of randomly selected targets are added to the list of hosts to test. If the binary entropy test failed, the address would be classified as 'Unknown' and would be passed to the same function that handled incremental addresses.

4. Next a host address is taken from the list of hosts to probe, and the process of probing is conducted.

5. Repeat from step 2 until the target host list has been exhausted or the maximum number of probes to transmit has been reached. The results of the simulation were then returned.

3.2.3.7 Algorithm classification

The host enumeration algorithms used in the research were designed to conduct search efforts either stochastically or deterministically. With the exception of the adaptive heuristic algorithm, any of the algorithms that involved random selection of targets was classified as stochastic (as indicated in Table 3.3). Similarly, and again with the exception of the adaptive heuristic algorithm, any algorithm that implemented a deterministic search pattern, was classified as deterministic (also shown in Table 3.3). The adaptive heuristic algorithm was excluded from such classifications since it employed both stochastic and deterministic search traits.

Table 3.3: Table of the algorithms designed for the study and their classification as stochastic or deterministic in nature.

Stochastic	Deterministic
MonteCarlo	Linear
GA	Stripe
Adaptive	Pattern

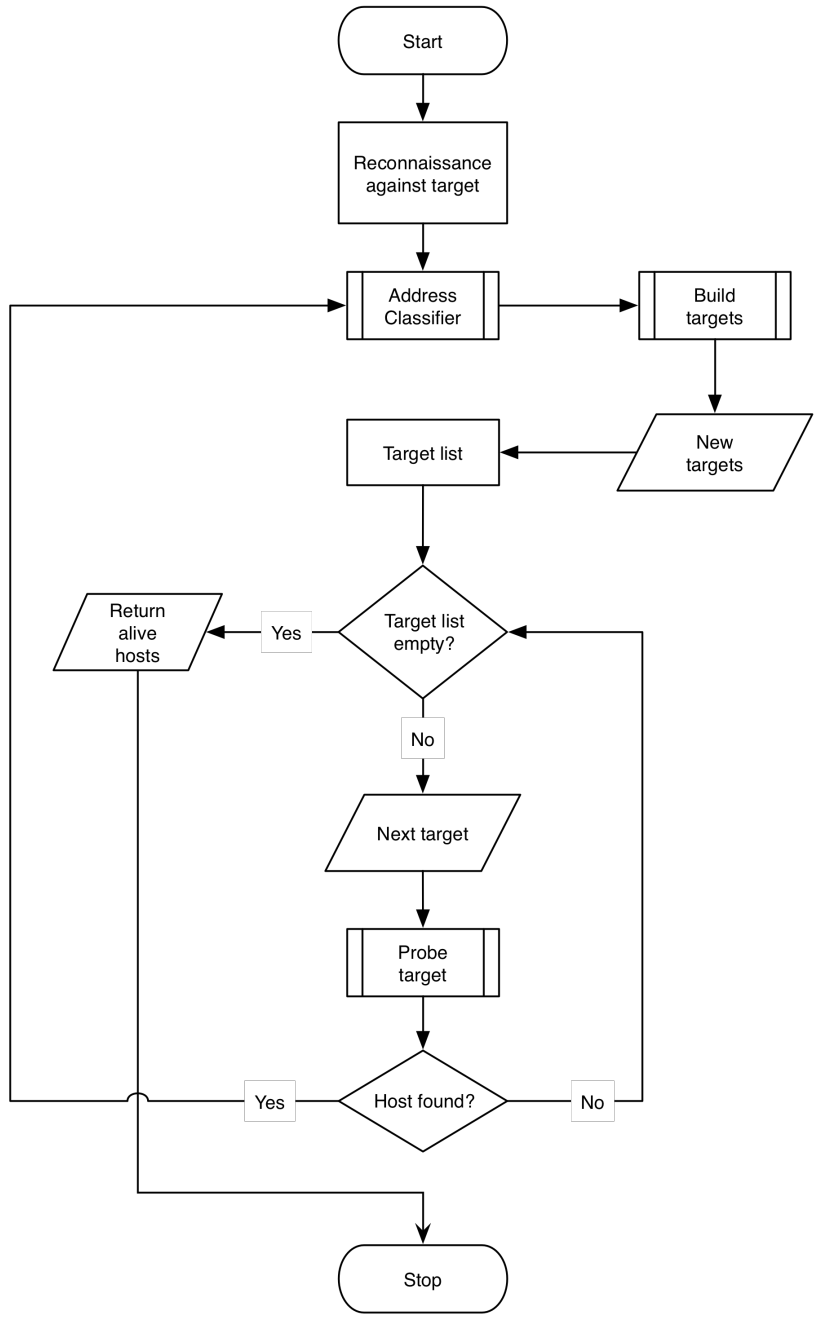
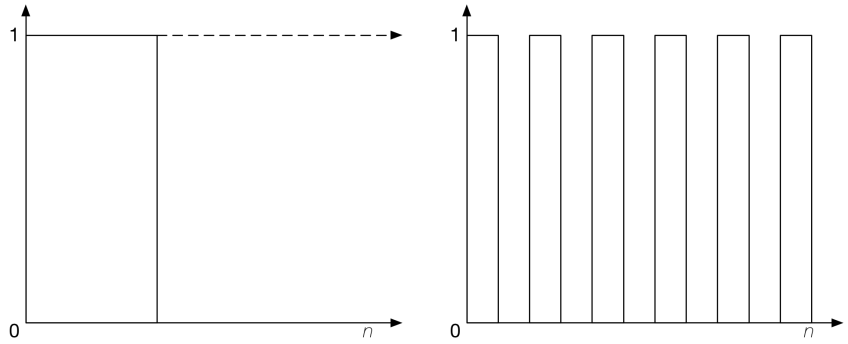
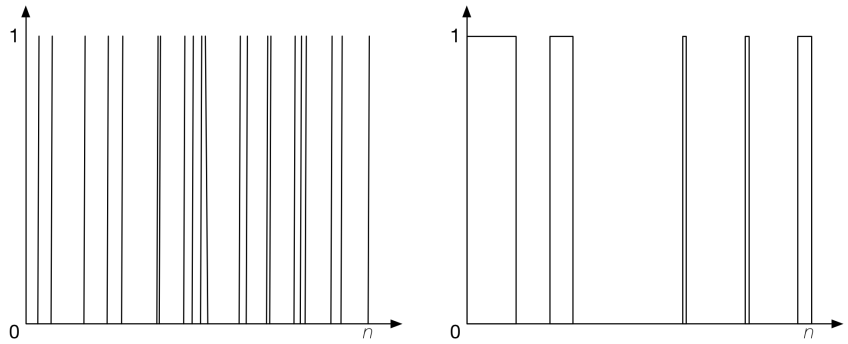


Figure 3-9: Flow chart depicting the major processes involved within the heuristic adaptive search algorithm that was designed for this research.



(a) Linear search algorithm. Searches the entire search space, or part-thereof sequentially

(b) Stripe search algorithm. Chunks the search space in the same fashion as a comb, and linear searches each chunk.



(c) Monte Carlo stochastic search algorithm. Randomly selects items in the search space to test.

(d) Adaptive search algorithm. Uses real-world IPv6 usage habits to influence search patterns.

Figure 3-10: Diagrammatic representation of hypothetical search operations for selected algorithms tested in the experiments. The y axes denote no delivered probe at $y = 0$ and a delivered probe at $y = 1$, the x axes represent the address space for the search operation.

3.2.4 Phase 3: Develop instrumentation and experiments, and perform pilot studies

Development and pilot testing occupied the majority of the time spent during the research process, and was revisited numerous times. The programs that were created to run the experimentation were developed using primarily open source tools. Each experiment consisted of a separate Python program which implemented the subject algorithm that was being measured. The standardised data formats for the results of the experiments were also developed in this phase. Where appropriate, validation of experimental techniques was performed and recorded.

Standardisation of the data consisted of combining the outputted pickled pandas

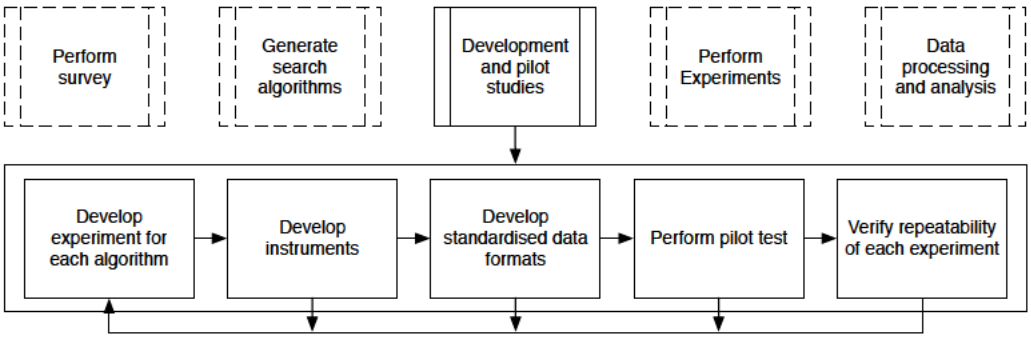


Figure 3-11: A breakdown of the Phase 3: Develop instrumentation and experiments, and perform pilot studies phase of the research. This phase involved integrating the search algorithms designed in Phase 2 into experimental programs, testing the outputs and conducting pilot tests.

dataframe file for each simulation into a single pandas dataframe. In doing so, all of the simulation data for a single experiment was available in one file. This dataframe was then serialised using the pickle library. The picked dataframe was then written to disk in an appropriate directory. Results were then analysed, and the findings included in Chapter 4 and explored further in Chapter 5.

During the course of the experiments, 64 bit unsigned integer values were used to represent IPv6 IIDs (as opposed to the commonly used colon-separated hexadecimal string representation). Since each discrete IPv6 IID represents a value between 0 and 2^{64}, representing the addresses as unsigned 64 bit unsigned integers did not negatively impact on the results. Representing the addresses in such a way simplified the operations of the experimental computer programs by eliminating unnecessary data type conversions. Complete IPv6 addresses, where necessary, were treated as 128 bit unsigned integer values within any relevant computer programs. The exception to this being where the IPv6 address data required visual observation. In these cases the IPv6 integer value was converted to the standard colon-separated hexadecimal string format for readability purposes.

3.2.4.1 Input data

The input data consisted of a number of files containing various data. The input files contained data that was either generated for the study, or gathered from the survey phase of the research. The primary input files used within the study were:

- `hex_patterns.txt`: This file contained newline separated, four character strings representing possible hexadecimal values used to construct hex words in the pattern-based heuristic algorithm.

- `ip_data_random.csv`: This CSV file contained a newline separated list of IPv6 addresses that were randomly generated by the author (Section 3.2.2 and Section 4.2 explores the process undertaken to generate these random addresses). This input file provided the initial dataset used within the randomised experiments (the `a` sub-experiments).

- `ip_data_surveyed_complete.csv`: The data contained in this CSV file consisted of the information gathered during the survey phase.

- `ip_data_surveyed.csv`: This CSV file contained the IPv6 addresses gathered in the survey. These data were taken from the `ip_data_surveyed_complete.csv` file by deleting the irrelevant columns (N.B. only the `ip6_address` column was retained in this file). This input file provided the initial dataset used within the surveyed experiments (the `b` sub-experiments).

3.2.4.2 Output data

In order to aid with the results analysis phase of the research, a standardised approach to outputting data were chosen for all experiments. Each experiment generated a number of output files, including result files and auxiliary metadata files. The result files served to store the results of individual simulations, and once collated, the experiment as a whole. The metadata files were used to store information about each experiment for documentation purposes. The output files created for each experiment included:

- A metadata file: This file was a plaintext file that included metadata surrounding the experiment. Metadata recorded included the input data filename used by the program, whether the algorithm was influenced by weights or purely stochastic, etc., all of the command line arguments used to execute the program (i.e. `sys.argv`) and all of the optional parameters used to modify the behaviour of the computer program (these were stored in a dictionary variable

named `options` in each experiment program). This file was used to ensure that important parameters of an experiment were recorded for repeatability.

- A series of pandas dataframes in pickled format: Pickling is the process of serialising a python object into a file that can then be deserialised to its original form. Every unique simulation stored a pickled dataframe on permanent storage upon completion of the simulation. These dataframe objects stored the data gathered from each simulation.

- A single pandas dataframe object in pickled format: Upon gathering data from each simulation in an experiment, the dataframes were combined back into a single entry to make analysis more convenient. This was performed during the standardisation of data subprocess of the data analysis phase (see Figure 3-1).

During the development phase of the study the research computers were configured to support the execution of the computer programs implementing the subject algorithms. Throughout this phase, the programs used to execute experiments were tested on the primary development computer, along with the compute cluster (outlined in Table 3.5). Figure 3-12 outlines the high level program design used in each experiment. The major variations of each experiment occurred in the Perform Experimentation phase (Section 3.2.5), where the variables pertaining to each subject algorithm are explored.

3.2.4.3 Materials and instrumentation

The Python programming language was chosen to develop the experiments, due to the flexibility Python offers for dealing with arbitrary data types, and also for the extensive third-party libraries available. Attention was paid to the programmatic realisation of the algorithms designed in Section 3.2.3. Third party libraries (such as `numpy` and `pandas`) which utilise lower level compiled languages were implemented in order to improve performance. Since Python uses hash tables for set and dictionary objects, membership tests are performed in constant time (i.e. $O(1)$) regardless of the number of elements in the table. Consequently, where possible, membership testing was performed against either set or dictionary objects.

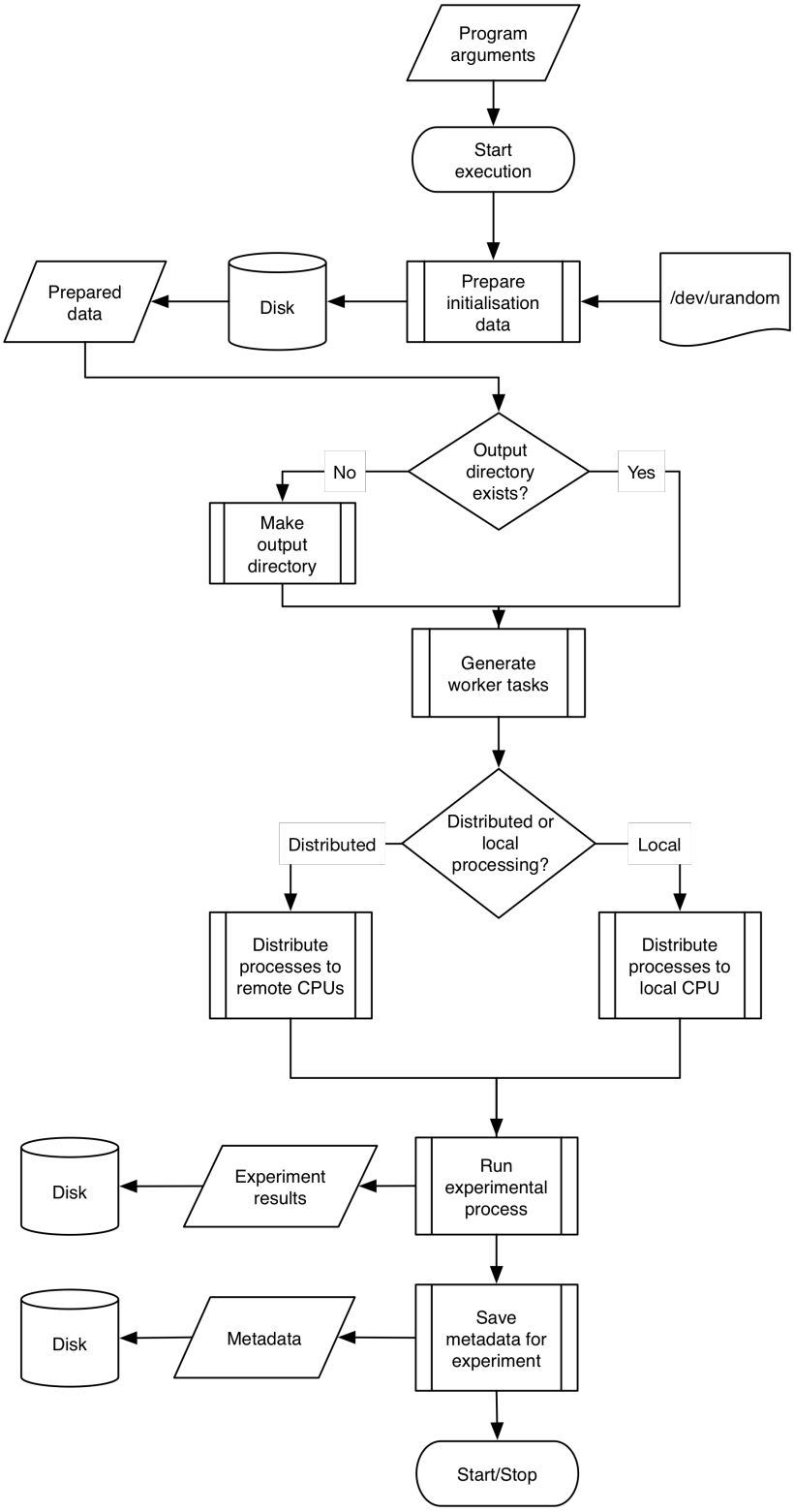


Figure 3-12: Flow diagram depicting the high level procedure that each experiment adhered to.

The computer program used to perform the search operations using the GA search algorithm represents an exception to the above statements in that it was written using the C programming language. This variation was required after conducting a pilot study using an equivalent program that was written using the Python programming language. The pilot study revealed that the Python-based GA program would not be able to complete sufficient enough generations in an appropriate timeframe for the research. It was measured that the Python implementation operated in $O(n^2)$ time. However, a C-based implementation maintained an average of $O(n)$ complexity, and was able to complete enough generations within the research timeframe to probe the maximum number of addresses (i.e. 2^{32} addresses).

Computer programs and third-party software libraries were used in the experiments. Instrumentation used to measure the changes in experiments predominantly used computer programs that made use of available open source software libraries. Common equipment and instruments were used throughout the study. The common materials and instruments used during the experiments are listed in Table 3.4.

Table 3.4: Table of associated software materials used for experimentation.

Name	Version	Description
Python2	2.7.6	The Python2 programming language
py2-ipaddress	2.0	Backport of the Python3 ipaddress library. Allows manipulation of string, hexadecimal or integer representation of IPv6 and IPv4 address numbers.
numpy	1.8.0	Performance driven numerical library for Python2 (McKinney, 2012; Walt, Colbert & Varoquaux, 2011)
pandas	0.13.1	Statistical analysis library for Python2. This library became the predominant analysis tool for recording, manipulating and reporting the data gathered in the study (McKinney, 2012)
scipy		Scientific programming toolkit for Python2. Contains numerous packages used for scientific computing including matplotlib, ipython, numpy and pandas (McKinney, 2010).
scoop		Python2 concurrent computing library. Enables tasks to be distributed across multiple computing resources (Hold-Geoffroy, Gagnon & Parizeau, 2014).
Python3	3.4.0	The Python3 programming language.
numpy	1.8.0	Performance driven numerical library for Python3 (McKinney, 2012; Walt, Colbert & Varoquaux, 2011)
pandas	0.13.1	Statistical analysis library for Python3 (McKinney, 2012)
scipy	0.13.3	Scientific programming toolkit for Python3. Contains numerous packages used for scientific computing including matplotlib, ipython, numpy and pandas (McKinney, 2010).
R-Studio	0.98.1102	(RStudio, 2014)
C programming language	clang-602.0.49	The C programming language.
gcc	Apple LLVM version 6.1.0	The gcc C/C+ compiler. Used to compile C code into executable binary files.
SimCList	1.5	A library that streamlines the creation and manipulation of linked lists in C.

Table 3.5: Table of associated hardware materials used for experimentation.

Name	Specifications	Description
Computer	• Supermicro H8QG6 • 4x AMD Opteron Processor 6274 (16 cores and 16 threads per CPU) = 64 CPU cores total • 16x Hyundai HMT31GR7BFR4C-H9 8Gib RAM = 128GiB RAM total	Computer used for development and testing experiments.
Retina MacBook Pro	• Model Name: MacBook Pro • Model Identifier: MacBookPro11,1 • Processor Name: Intel Core i7 • Processor Speed: 2.8 GHz • Number of Processors: 1 • Total Number of Cores: 2 • Memory: 16 GB	Primary computer used for development, quality testing and documenting results.
Compute cluster	• 534x 32 bit CPUs • 135x Computer hosts	Primary computers used for experiments

During the perform experiments phase (phase 4) of the research, all of the Python-based computer programs used to implement the subject algorithms were initiated in the same way. The computer program `screen` was used to create a persistent terminal session with a common name for the experiment. Then, the Python module `scoop` was executed from a command line interface, to initialise the scalable computing program. The experiment Python program and all associated arguments, were provided to `scoop` as an argument, and would be launched accordingly. The `screen` session was then detached to allow the process to continue execution in the background. During the development phase, where multiprocessing capabilities were not required, the `scoop` program was not always utilised. In such situations, the Python program would be executed directly with its associated arguments. For C-based computer programs, a similar process was observed. The computer program `screen` was used to create a persistent terminal session with a common name for the experiment. A Python program was used to execute the C program the required number of times with the parameters for the subject experiment. This Python program managed the distribution of processes across the available compute resources. The executed C program then ran as a single process.

Table 3.6: Table of instrumentation used throughout the study.

Name	Version	Description
pandas	0.13.1	Pandas library for Python2. Provides data storage and analysis capabilities (McKinney, 2012).
random	2.7.6	Python2 built in random module. Provides pseudo-random number generation capabilities.
time	2.7.6	Python2 built in time module. Provides access to system clock for timing tasks. Used to measure the process time of experiments.
numpy.random	1.8.0	Numpy's random module. Provides pseudo-random number generation capabilities. (McKinney, 2012; Walt, Colbert & Varoquaux, 2011)
ipython notebook	2.0.0	IPython Notebook provides a persistent journal of Python activities (Pérez & Granger, 2007). The notebooks were used for pilot studies, as well as validation of experimental procedures and results.
matplotlib	1.3.1	Graphing library for Python. Used to create plots out of data and results (Hunter, 2007).
scipy	0.13.3	Scientific programming toolkit for Python. Contains numerous packages used for scientific computing including matplotlib, ipython, numpy and pandas (McKinney, 2010).

Table 3.7: Table displaying the computer programs and auxiliary libraries created for the experiments and utilised throughout the study.

Experiment	Program	Additional libraries used
Adaptive search algorithm	`heuristic.py`	• `ann.py` • `ip6mangle.py`
Genetic Algorithm	`ga`	• N/A
Linear search algorithm	`linear.py`	• `ip6mangle.py`
Monte Carlo search algorithm	`monte_carlo.py`	• `ip6mangle.py`
Pattern-based search algorithm	`pattern.py`	• `ip6mangle.py`
Stripe search algorithm	`stripe.py`	• `ip6mangle.py`

3.2.4.4 Pilot studies

Throughout the development cycle, and at the completion of the development of an experiment, pilot tests were performed. Pilot testing ensured that the algorithm and associated computer programs functioned as expected. Additionally, the results of the pilot test served to ensure that sufficient data were being recorded and that the experiments could be repeated.

Repeatability was ensured by running the experiment multiple times using the seed values from the pilot test, and verifying the results were the same. This ensured that not only could experiments be repeated, but also that if an experiment failed to conclude, it could be resumed from a known state (reducing the overall time to completion).

Each pilot study tested the subject algorithm's delivery of 10,000 probes in a single

simulation, and recorded each probed address. The recorded addresses ensured the distribution of probes across the address space reflected the expectations for the algorithm and was able to be plotted. The results of the final pilot studies for each experiment have been included in their relevant sections in Chapter 4.

3.2.5 Phase 4: Perform experimentation

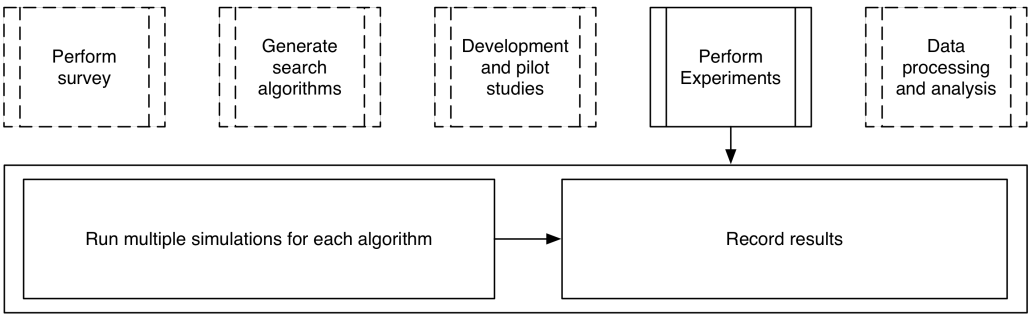


Figure 3-13: A breakdown of the the Phase 4: Perform experimentation phase of research phase including major processes involved.

In this phase of the research the experiments were conducted and the results recorded. A computer simulation method was used to test and measure the subject algorithms. Each experiment that was conducted involved applying the subject algorithm against two datasets separately. The experiments were conducted using the randomly generated IPv6 addresses as the valid target nodes once, and then again using the surveyed IPv6 addresses as the valid host nodes. In this fashion each algorithm was tested against an unpredictable set of targets, as well as against an emulation of a real world IPv6 network.

Although the algorithms varied, the computer programs used in the research followed a common format, and stored information in a common fashion (see Figure 3-12). Each program commenced by parsing command line input parameters and establishing global variables. During the qualified experiments concurrent computing means were utilised, in which case the broker process would distribute tasks to worker processes on other computers (Hold-Geoffroy, Gagnon & Parizeau, 2014). Each experiment therefore utilised a number of processors to complete. Simulations were queued and pushed to available compute resources on the parallel computing cluster for execution.

A number of variables were considered as constraints to the operations of the independent algorithms. The maximum total number of probes that it was deemed acceptable or permissible for any algorithm to send was capped at 2^{32}. This number was chosen since it represents the entirety of the IPv4 address space, which is viewed as feasible to perform host discovery against (Graham, 2013c). Additionally, with the exception of experiments applying the stripe search algorithm, the IPv6 host address space (2^{64} out of 2^{128}) was partitioned into "buckets", "bins" or "chunks" (these terms are used interchangeably throughout this document) representing 2^{32} buckets containing 2^{32} unique addresses per bucket. The stripe search required the use of larger bins (sized at 2^{48} addresses per bucket or 2^{16} buckets total), in order to allow for even distribution of probes across the entire space, whilst still adhering to the 2^{32} limit on transmitted probes. Without adjusting the bucket size to suit, the stripe search algorithm would only probe one address per bucket in the 64 bit address space. By using 2^{48} rather than 2^{32} as the bucket size, the algorithm could probe 2^{16} addresses per bucket.

Using the data collected during the survey phase, weights were applied to each bucket in the IPv6 address space based upon the number of collected IIDs that were recorded for that bucket. The weighting calculation is expressed as $w = \forall n \in k : \frac{n}{t}$, where w represents the weights for each bucket, n represents the number of samples in the current bucket, k represents all of the buckets and t represents the total number of samples. This process was also applied to the randomly generated dataset to generate weights for that dataset.

Each experiment involved performing repeated simulations under controlled conditions. This served to provide large enough samples of results for each algorithm to account for the stochastic nature of experimental parameters. Once a simulation was completed, the results of the simulation were recorded. This consisted of the results being added to a `pandas` dataframe and then pickled and committed to a secondary storage device (i.e. a hard disk drive) as described in Section 3.2.4.2. Once the final simulation in the experiment was completed, a metadata file with the experimental parameters was committed to secondary storage, and the experiment was concluded.

Each experiment also consisted of two sub-experiments (sub-experiment `a` and sub-

experiment `b`) which modified the target dataset that the algorithm was tested against. The sub-experiments labeled `a` used the randomly generated dataset of IPv6 addresses for search operations and to generate weightings, whilst `b` experiments were tested against the surveyed dataset of IPv6 addresses. All experimental conditions were constant between the `a` and `b` sub-experiments other than the dataset that the subject algorithm was tested against.

3.2.5.1 Linear search algorithm

The linear search algorithm was the subject of three independent experiments. For each experiment, the sub-experiments `a` and `b` tested the subject algorithm against the randomised and surveyed datasets, 100 times respectively.

1. The first experiments were conducted with the linear search algorithm starting its search operation from the first discrete address in the target space (i.e. 0) and then continuing to search each address until the maximum number of transmitted probes was reached.

2. The second experiments started the search operation at a randomly chosen address in the target address space, with the caveat that: $start\_point + maximum\_probes < 2^{64}$ or else $start\_point = 2^{64} - maximum\_probes$.

3. Finally, the third experiments used a weighted random choice to select a start point in the address space, based upon the weighting calculation explained in Section 3.2.5. Again this start point was required to adhere to the above constraint, ensuring that the start and end points lay within the bounds of the target address space.

The experimental parameters for the linear search algorithm's experiments are detailed in Table 3.8.

3.2.5.2 Stripe search algorithm

Next, the stripe search algorithm was tested. Stripe search was the subject of two distinct experiments. For each experiment, the sub-experiments `a` and `b` tested the subject algorithm against the randomised and surveyed datasets, respectively, 100 times.

Table 3.8: Summary of the conditions and parameters influencing each of the linear search algorithm's experiments.

Experimental parameters	Experiment number		
	1	2	3
Experiment codename	ZeroOrigin	Random	WeightedRandom
Valid address datasets tested	• 1a) Random dataset • 1b) Surveyed dataset	• 2a) Random dataset • 2b) Surveyed dataset	• 3a) Random dataset • 3b) Surveyed dataset
Simulations per dataset	100	100	100
Start address selection method	0	Randomly selected	Selected using weighted random choice
Max transmitted probes	4294967296	4294967296	4294967296
Number of bins (2^n)	32	32	32
Addresses per bin (2^n)	32	32	32
PRNG seed	32 bytes extracted from `/dev/urandom`	32 bytes extracted from `/dev/urandom`	32 bytes extracted from `/dev/urandom`

A parameter was used to signify the point where the algorithm would originate and conclude the search operation in each bin. It is important to reiterate that the size of each bin that was chosen for the stripe search algorithm differs from the standard size used in other experiments. This decision was made in order to keep the maximum number of transmitted probes to 2^{32}, and still probe a reasonable number of hosts per interval. Without modifying the size of the buckets, the stripe search algorithm would only probe one host per interval.

In the first experiment, the algorithm's starting point was randomly generated such that $start\_point + probes\_per\_interval < 2^{bin\_size}$ or else $start\_point = 2^{bin\_size} - probes\_per\_interval$. Experiment 2 assigned a starting point of 0 to each simulation that was conducted. The experimental parameters for the stripe search algorithm's experiments are detailed in Table 3.9.

3.2.5.3 Monte Carlo search algorithm

The Monte Carlo search algorithm was used in two sets of experiments. For each experiment, the sub-experiments `a` and `b` tested the subject algorithm against the randomised and surveyed datasets, 100 times respectively.

In the first set of experiments every target was pseudorandomly selected across the entire space. In the second set of experiments, the address space was binned, and a weighted choice was used to select candidate bins. Target addresses were chosen

Table 3.9: Summary of the conditions and parameters influencing each of the stripe search algorithm's experiments.

Experimental parameters	Experiment number	
	1	2
Experiment codename	Random	ZeroOrigin
Valid address datasets tested	• 1a) Random dataset • 1b) Surveyed dataset	• 2a) Random dataset • 2b) Surveyed dataset
Simulations per dataset	100	100
Start address selection method	Randomly selected	0
Max transmitted probes	4294967296	4294967296
Transmitted probes per bin	65536	65536
Number of bins (2^n)	16	16
Addresses per bin (2^n)	48	48
PRNG seed	32 bytes extracted from `/dev/urandom`	32 bytes extracted from `/dev/urandom`

pseudorandomly from the selected bin. The experimental parameters for the Monte Carlo search algorithm's experiments are detailed in Table 3.10.

3.2.5.4 Genetic algorithm

The GA experiments tested the genetic algorithm against the surveyed and randomised datasets during three discrete experiments. The first experiment tested the GA against the IPv6 address datasets using an initial population of 0's. The second experiment applied the GA to the IPv6 address datasets using an initial population that was generated pseudorandomly. Finally the third experiment applied the genetic algorithm to the IPv6 address datasets using an initial population that was generated by using a weighted random choice based upon the frequency of buckets in the dataset.

The experimental parameters for the GA search algorithm's experiments are detailed in Table 3.11.

3.2.5.5 Pattern-based heuristic search algorithm

Next, experiments were conducted that tested the pattern-based algorithm against the target dataset. In these experiments, each sub-experiment `a` and `b` tested the subject algorithm against the randomised and surveyed datasets, respectively, 100 times.

The pattern-based algorithm generated the target list using a variety of approaches, firstly searching through address `::` to `::100:ffff` to cover low range incremental

Table 3.10: Summary of the conditions and parameters influencing each of the Monte Carlo search algorithm's experiments.

Experimental parameters	Experiment number	
	1	2
Experiment codename	Random	WeightedRandom
Valid address datasets tested	• 1a) Random dataset • 1b) Surveyed dataset	• 2a) Random dataset • 2b) Surveyed dataset
Simulations per dataset	100	100
Target address selection method	Randomly selected	Selected using weighted random choice
Max transmitted probes	4294967296	4294967296
Number of bins (2^n)	32	32
Addresses per bin (2^n)	32	32
Number of bins to search	All	1000
PRNG seed	32 bytes extracted from `/dev/urandom`	32 bytes extracted from `/dev/urandom`

Table 3.11: Summary of the conditions and parameters influencing each of the GA search algorithm's experiments.

Experimental parameters	Experiment number		
	1	2	3
Experiment codename	ZeroOrigin	Random	WeightedRandom
Valid address datasets tested	• 1a) Random dataset • 1b) Surveyed dataset	• 2a) Random dataset • 2b) Surveyed dataset	• 3a) Random dataset • 3b) Surveyed dataset
Simulations per dataset	100	100	100
Initial population selection method	Initialised to 0	Randomly selected	Selected using weighted random choice
Max transmitted probes	4294967296	4294967296	4294967296
Mutation rate	2%	2%	2%
Number of bins (2^n)	32	32	32
Addresses per bin (2^n)	32	32	32
Number of Generations to breed	100000001	100000001	100000001
Number of Organisms per Generation	100	100	100
Number of bins to search	All	All	All
PRNG seed	4 bytes extracted from `arc4random()`	4 bytes extracted from `arc4random()`	4 extracted from `arc4random()`

addresses.

Next, IPv4-in-IPv6 style addresses were checked. The algorithm that was used checked every IPv4-in-IPv6 address between `::1:0:0:0` and `::255:255:255:255`. The addresses between `::` and `::255:255:255` were not considered for testing through this method for three reasons. Firstly, some of the addresses overlap with the low range

linear search performed previously by the algorithm. Secondly, the IPv4 addresses between `0.0.0.0` and `0.255.255.255` (`0.0.0.0/8`) are reserved by IANA and have not been made available for public usage (IANA, 2014a). The act of embedding an IPv4 address into an IPv6 address is a convenience method for administrators to aid in transitioning to the new protocol. It is therefore unlikely that they would translate addresses that are not allowed to be registered. Finally, and most tellingly, without restricting the number of probes transmitted by this method, it would exceed the maximum number of transmitted probes. Since IPv4 is a 32 bit address space, testing the entire IPv4-in-IPv6 space would still exhaust 2^{32} probes, which is the maximum number of probes an experiment was allowed to deliver. It was therefore pertinent to limit this parameter.

Next, a list of four character hexadecimal words were permuted into 16 character words, converted from hexadecimal to integer representation, and then probed. The starting list contained nine unique four character words, which were combined to form 17,160 unique IPv6 IIDs.

The experimental parameters for the pattern-based search algorithm's experiments are detailed in Table 3.10.

3.2.5.6 Adaptive heuristic search algorithm

Only a single experiment was conducted using the adaptive algorithm. The subject algorithm was tested against the randomised (sub-experiment `a`) and surveyed datasets (sub-experiment `b`) respectively, 100 times. The adaptive algorithm used a classification system to determine a course of action, after the successful probing of an address.

- For addresses that were classified as 'EUI-64' a range of addresses were constructed within the same MAC address OUI scope in such a way that

$$start\_point = n - \frac{increment\_range}{2} \tag{3.1}$$

and

$$end\_point = n + \frac{increment\_range}{2} \tag{3.2}$$

Table 3.12: Summary of the conditions and parameters influencing each of the pattern-based heuristic search algorithm's experiments.

Experimental parameters	Experiment number
	1
Experiment codename	Pattern
Valid address datasets tested	• 1a) Random dataset • 1b) Surveyed dataset
Simulations per dataset	100
Start address selection method	
Max transmitted probes	4294967296
Number of bins (2^n)	32
Addresses per bin (2^n)	32
Wordy address components	• `0xcafe` • `0xdead` • `0xface` • `0xbeef` • `0xb00b` • `0x0000` • `0xffff` • `0x1337` • `0x0dad`
Total wordy addresses constructed	17160
PRNG seed	32 bytes extracted from `/dev/urandom`

while

$$end\_point < max \tag{3.3}$$

else

$$start\_point = 2^{24} - increment\_range \tag{3.4}$$

and

$$end\_point = max - 1 \tag{3.5}$$

where n is the current target and $max = 2^{24}$.

- If an address was classified as 'Incremental' then a range calculation was used such that

$$start\_point = n - \frac{increment\_range}{2} \tag{3.6}$$

and

$$end\_point = n + \frac{increment\_range}{2} \tag{3.7}$$

while

$$end\_point < max \tag{3.8}$$

else

$$start\_point = 2^{64} - increment\_range \tag{3.9}$$

and

$$end\_point = max - 1 \tag{3.10}$$

where n is the current target and $max = 2^{64}$.

- Finally, if an address was classified as 'Stochastic' a binary entropy test was conducted to test the efficacy of the classification. If the binary entropy test failed (i.e. it returned a result less than 0.7), the classification was determined to be false and the target classified as "Unknown". On the other hand, if the test passed (i.e. the results were greater than or equal to 0.7), 65536 (i.e. the value of $increment\_range$) random targets were added to the target list for probing.

The experimental parameters for the Adaptive search algorithm's experiments are detailed in Table 3.10.

3.2.6 Phase 5: Data processing and analysis

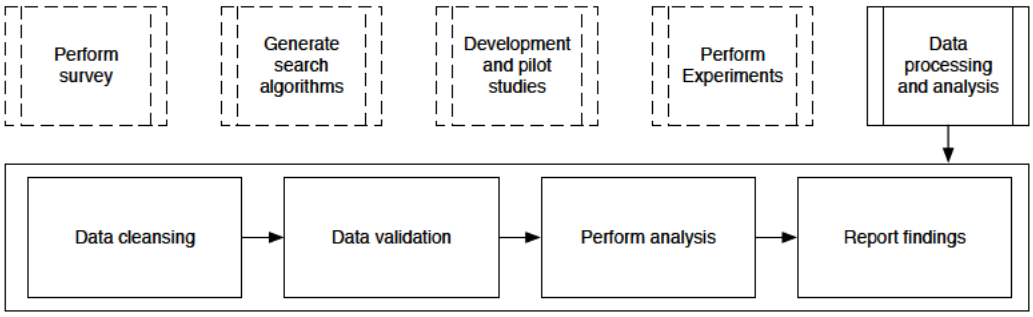


Figure 3-14: A breakdown of the the Phase 5: Data processing and analysis phase of research phase including major processes involved.

Table 3.13: Summary of the conditions and parameters influencing each of the Adaptive heuristic search algorithm's experiments.

Experimental parameters	Experiment number
	1
Experiment codename	Adaptive
Valid address datasets tested	• 1a) Random dataset • 1b) Surveyed dataset
Simulations per dataset	100
Start address selection method	
Max transmitted probes	4294967296
Number of bins (2^n)	32
Addresses per bin (2^n)	32
Additional range of addresses to increment	65536
Number of "public" records gathered	100
PRNG seed	32 bytes extracted from `/dev/urandom`

3.2.6.1 Data cleansing and validation

The data that were collected from the experimental phase of the research were cleansed and validated prior to undertaking analysis. The raw results collected from the experimentation process included a number of superfluous attributes (such as the recorded PRNG seed and state values) these attributes were removed from the analysis dataset.

The data that were collected were validated to confirm that the data was correct for analysis. This involved performing type checks on the data. Since the data used large integers to represent IPv6 addresses (i.e. greater than 64 bit integers), overflow checking was performed to ensure that the positive integers were preserved correctly by the data storage and analysis program.

3.2.6.2 Data analysis techniques

In order to analyse the data each sub-experiment was aggregated. This meant that the individual simulations in a sub-experiment had their results aggregated to provide a macro summary of the sub-experiment. Analysis of the data were conducted using the `pandas` Python library, and the `RStudio` statistical analysis program. The `pandas` library provided interfaces to conduct descriptive statistics on dataframes. These interfaces were used to provide descriptive statistics on each aggregated sub-experiment. The `RStudio` computer program was used to analyse the dataset and conduct inferen-

tial statistics.

The anticipated results of the experiments was primarily count data, representing the number of successful probes delivered per simulation. Due to the nature of the problem it was anticipated that the results would contain data that does not conform to normal distribution, and that there would be the presence of zeroes in the data. Because of these qualities of the expected results descriptive statistics were employed, along with non-parametric inferential statistics. In particular the descriptive statistics used included the mean, median, maximum and minimum values for the simulations conducted in each sub-experiment.

For each of the primary research hypotheses inferential statistics were used. The data were tested for normality using the Shapiro-Wilks test for normality, as is recommended by Montgomery (2009). Data that were discovered to be normally distributed would be tested using the one-sample and independent samples t-tests to test the research hypotheses (Montgomery, 2009). If the data were discovered to be non-parametric the Wilcoxon tests, including the ranked-signed test and the ranked-sum test, were considered to be appropriate. This decision was made because the Wilcoxon tests rely upon ranking of data, rather than a direct comparison of means (Montgomery, 2009; Corder & Foreman, 2011). All statistical testing for the study was performed at the 95% confidence interval (i.e. $\alpha = 0.05$). In practice, the 95% confidence interval means that a type I error (where the results are incorrectly classified as significant), may be committed five times out of 100 (Sheskin, 2000).

3.3 Ethical considerations

This research did not involve any human or animal subjects, nor any private or identifying data about computer systems or their users. The data gathered during the experiment was fabricated by the author and gathered from real world, publicly accessible sources.

Ethical approval for the orchestration of this research project was obtained from the Edith Cowan University's Ethics Approval board (valid between the 9th of January, 2012 until the 9th of January, 2015). All research undertaken in this study was done

strictly in accordance with the ethics policy governing the approval.

Chapter 4

Results

4.1 Surveyed dataset

The survey phase provided examples of IPv6 addresses being used in public networks globally. These addresses served to form one of the two target networks that were searched during the off-link IPv6 host enumeration experiments. The resulting dataset of IPv6 addresses arising from the survey is referred to as the 'surveyed dataset'. Any figures that refer to the surveyed dataset are coloured in green. The results observed in this phase of the research have been published in Carpene and Woodward (2014). The survey was conducted by enumerating DNS servers for potential IPv6-enabled hosts. Table 4.1 outlines the sources for the hostnames that were queried during the DNS enumeration, along with the number of records each contributed to the IPv6 address dataset. In total, 216,512 unique IPv6 addresses were gathered during the survey phase.

After the survey data was gathered some analysis was performed. During analysis the data were transformed. First, the complete IPv6 addresses were converted into integer values. Next the IPv6 IIDs were extracted (i.e. the 64 bit network prefix was removed) by converting the string representation of the IPv6 address into an integer, and then applying:

$$IID = IntegerIPv6Address \mod 2^{64} \tag{4.1}$$

Of the 216,512 unique IPv6 addresses gathered, a total of 41,171 unique IPv6 IIDs

were extracted. Table 4.2 shows the unique IPv6 IIDs that were gathered from each of the survey sources. It was apparent that the majority of IIDs collected in the survey were contained to the first 32 bits of the 64 bit address space (120,898 out of 216,512 total observed IIDs). This can be seen in Figure 4-1. There was an even distribution of addresses over the remainder of the address space (Figure 4-1).

After classifying the surveyed data using the ANN classification system described in Carpene et al. (IN PRESS), it was observed that approximately half of the sampled IIDs conformed to the incremental IID construction type (displayed in Table 4.3). It was also clear that the incremental IID category contained few unique records. Only 12.71% of the observed Incremental/Manually assigned IIDs were unique. This indicates that address reuse is occurring in public IPv6 networks. It was also observed that the stochastically generated IID category exhibited almost the same amount of unique IIDs as the Incremental IID category (18473 and 20338 respectively). These results and further analysis have been conducted in Carpene and Woodward (2014).

Table 4.1: Information sources used to provide domain names as the foundation of the survey. Against each source are the total IPv6-enabled subdomains and IPv6 addresses extracted from the survey.

Source name	Unique IPv6 addresses gathered	Unique IPv6 enabled subdomains discovered	Description
Alexa Top one million websites	151360	109276	The Alexa company posts a list of the top one million websites daily. Each subdomain in this list was queried and the results collected (Alexa Internet, Inc., 2014).
Unknown	61116	46063	These addresses were collected from sources that were unrecorded at the time the survey was conducted.
IPv6-Hosts list	4032	2176	A public information source providing a list of commonly used internet services and their IPv6 addresses (xslidian & VersusClyne, 2014).
Australian Government websites	4	4	List of Australian government subdomains (Australia.gov.au, n.d.).
Total	216512	157519	

Table 4.2: The results from the IPv6 survey revealing the number of unique IPv6 IIDs gathered from each information source.

Source name	IIDs Observed
Alexa Top one million websites	29231
Unknown	16560
IPv6-Hosts list	297
Australian Government websites	4
Total unique IIDs collected	41171
Total observed IIDs	216512

Table 4.3: Observed IPv6 addresses from the surveyed dataset were classified using an ANN classification system by their construction type.

Classification type	Unique IIDs Observed	All observed IIDs (including duplicates)
Incremental/Manually Assigned	20338	159939
Stochastic	18473	49967
EUI-64	2360	6606

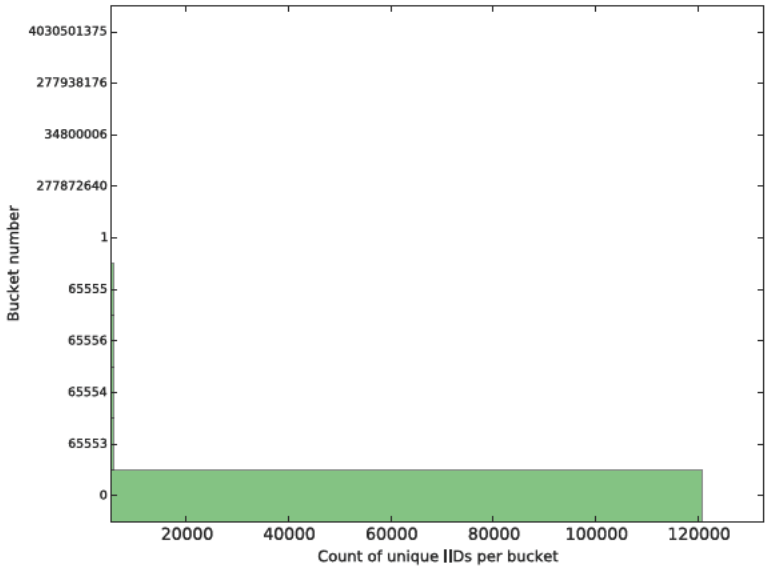


Figure 4-1: Chart of IID frequency distribution observed from the IPv6 survey chunked into 2^{32} bins with the top 10 observed bins displayed. The y axis represents the top ten observed bin numbers (between 0 and 2^{32}) whilst the x axis represents the frequency of IIDs per bin.

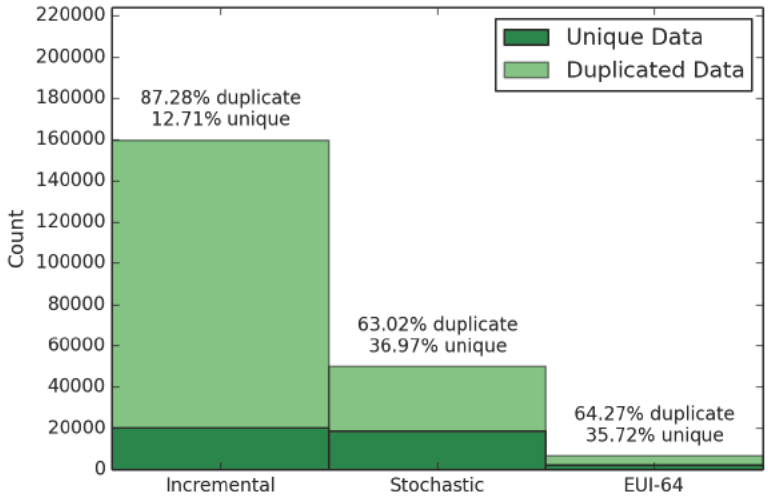


Figure 4-2: IID data collected in the survey was classified using an ANN classification system (Carpene, Johnstone & Woodward, IN PRESS) into their relevant construction types. The x axis denotes the classification category, whilst the y axis shows the frequency of observed IIDs in each category. The subplots depict the number of unique IIDs and the total number of IIDs observed in each construction category respectively. The percentage of duplicate and unique IIDs classified for each category is included at the head of each bar.

4.2 Randomised dataset

The randomised dataset was constructed using Python's Mersenne Twister-based PRNG to create 50,000 unique values between 0 and 2^{64}. These addresses served to form one of the two target networks that were searched during the off-link IPv6 host enumeration experiments. The process was as follows:

1. Generate 50,000 random address values.
2. Select a random number of addresses from the dataset (between 1 and 1000).
3. Duplicate the selected address a random number of times (between 1 and 2000).

The results for the pseudorandomly generated 'randomised dataset' are included in Table 4.4. Any figures that refer to the randomised dataset are coloured in blue. Figure 4-3 displays the distribution of the randomly generated IIDs across the top ten observed bin numbers. The distribution of addresses over the top ten bins can be seen to be almost even.

As with the surveyed dataset, the randomly generated dataset was classified into three IID construction categories; Incremental/Manually assigned, EUI-64 and Stochastic IID construction. Table 4.5 displays the results of the classification process. It has been seen that unlike the surveyed dataset, the majority of the addresses constructed were classified as stochastically generated. The least observed address category for the randomised dataset was the incrementally or manually assigned addresses.

Table 4.4: The results from the randomly generated IPv6 dataset. The number of unique IIDs that were generated, as well as the total number (including duplicates) are included.

Source name	IIDs generated
Unique IIDs generated	50000
Total observed IIDs	482812

Table 4.5: Observed IPv6 addresses from the randomised dataset classified using an ANN classification system by their construction type.

Classification type	Unique IIDs Observed	All observed IIDs (including duplicates)
Stochastic	38401	356360
EUI-64	7128	72896
Incremental/Manually Assigned	4471	53556

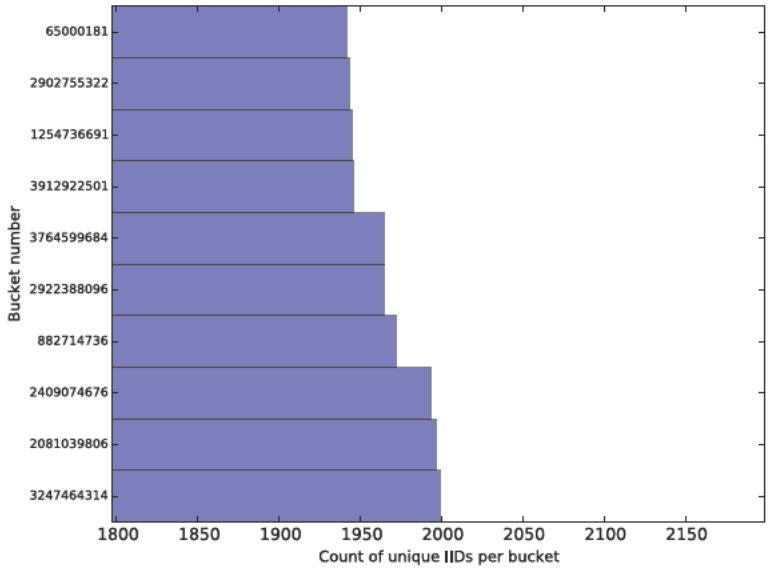


Figure 4-3: Chart of IID frequency distribution observed from the pseudorandom IPv6 generation process chunked into 2^{32} bins with the top 10 observed bins displayed. The y axis represents the top ten observed bin numbers (between 0 and 2^{32}) whilst the x axis represents the frequency of IIDs per bin.

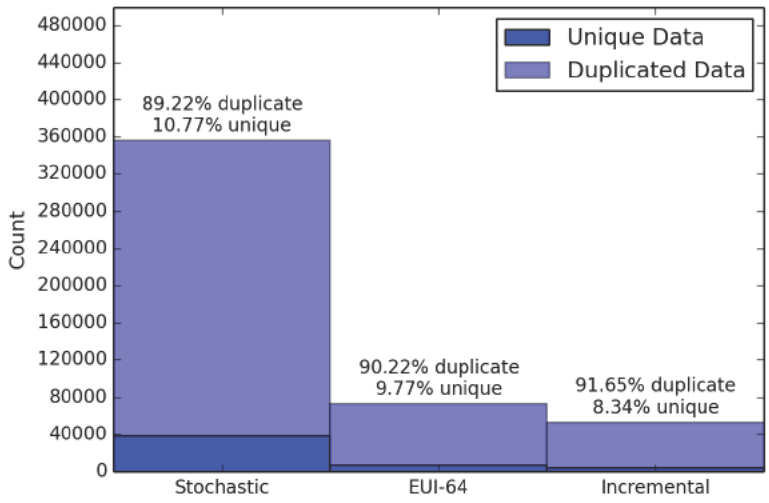


Figure 4-4: IID data generated through the randomised process was classified using an ANN classification system (Carpene, Johnstone & Woodward, IN PRESS) into their relevant construction types. The x axis denotes the classification category, whilst the y axis shows the frequency of observed IIDs in each category. The subplots depict the number of unique IIDs and the total number of IIDs observed in each construction category respectively. The percentage of duplicate and unique IIDs classified for each category is included at the head of each bar.

4.3 Linear search

The linear search algorithm was designed to reflect the sequential search technique that is commonly employed against IPv4 networks and resources. The experiment was designed to test the efficacy of the algorithm within the IPv6 protocol's address space. The algorithm was the subject of three main experiments; the `Linear-1` experiment, which tested the algorithm with the starting point of the sequential search at the first address in the address space (i.e. address `::`); the `Linear-2`, which saw the algorithm commence search operations at randomly generated starting points within the address space; and finally the `Linear-3` experiments, which saw the algorithm commence search operations at starting points in the address space that were chosen using a weighted random choice operation.

The Python program `linear.py` was used to implement the linear search algorithm. The program was configured to deliver a maximum of 2^{32} probes, per simulation, to IPv6 addresses sequentially from a designated starting address. The starting addresses varied per experiment to align with the specifications detailed above. The program was also configured to perform 100 simulations for each sub-experiment. The outcomes of these experiments are detailed below.

4.3.1 Experiment Linear-1: Zero-origin linear search

The zero-origin `Linear-1` experiments tested the linear search algorithm against the randomised and surveyed datasets respectively. This experiment differentiated from the other experiments involving the linear search algorithm by specifying a starting point of address 0 (`::`) and searching sequentially until the conclusion point of address 4294967295 (`::ffff:ffff`) for each simulation conducted. As a result, for the sub-experiments `Linear-1a` and `Linear-1b` the `linear.py` program was configured to run 100 simulations each applying the linear search algorithm to the subject dataset with the starting address for each simulation's search operation set to 0. Consequently, the `Linear-1` experiment was deterministic and completely controlled.

A pilot study was performed prior to commencing data collection for the `Linear-1` experiment to test the linear search algorithm under experimental conditions. The pi-

lot study tested the `linear.py` program and recorded the probing of 10,000 addresses under the `Linear-1` experimental conditions (defined in Chapter 3). A graph displaying the distribution of the recorded probes from the `Linear-1` pilot study across the address space is included in Figure 4-5. This graph provides an approximation for how each simulation for the linear search algorithm probed target addresses during the `Linear-1` sub-experiments. From this figure it is evident that the algorithm probed the addresses 0 (`::`) to 10000 (`::2710`) sequentially in linear time as expected. The pilot study's probe distribution aligned with the projected distribution for the linear search algorithm.

The results for the zero-origin `Linear-1a` and `Linear-1b` sub-experiments are included in Table 4.6. The `Linear-1` experiment recorded the maximum, and minimum mean number of successful probes for its sub-experiments across all of the experiments conducted, at 16,284 and 0 successful probes for `Linear-1b` (see Figure 4-7(a)) and `Linear-1a` (see Figure 4-6(a)) respectively.

Table 4.6: The descriptive statistics of the results for the Zero Origin `Linear-1` experiment (sub-experiments `Linear-1a` and `Linear-1b`).

Experimental parameters		Experiment number	
		Linear-1a	Linear-1b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	16284 (39.5521%)
	Mean	0.00 (0.0000%)	16284.00 (39.5521%)
	Minimum	0 (0.0000%)	16284 (39.5521%)
	Median	0.0	16284.0
	Total unique hosts discovered	0	16284
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	28049.57	53731.63
	Mean	23758.54	50441.50
	Minimum	6132.67	11165.95

As mentioned above, the `Linear-1b` sub-experiment successfully probed an average of 16,284 valid hosts during the search operations. These results represent successful

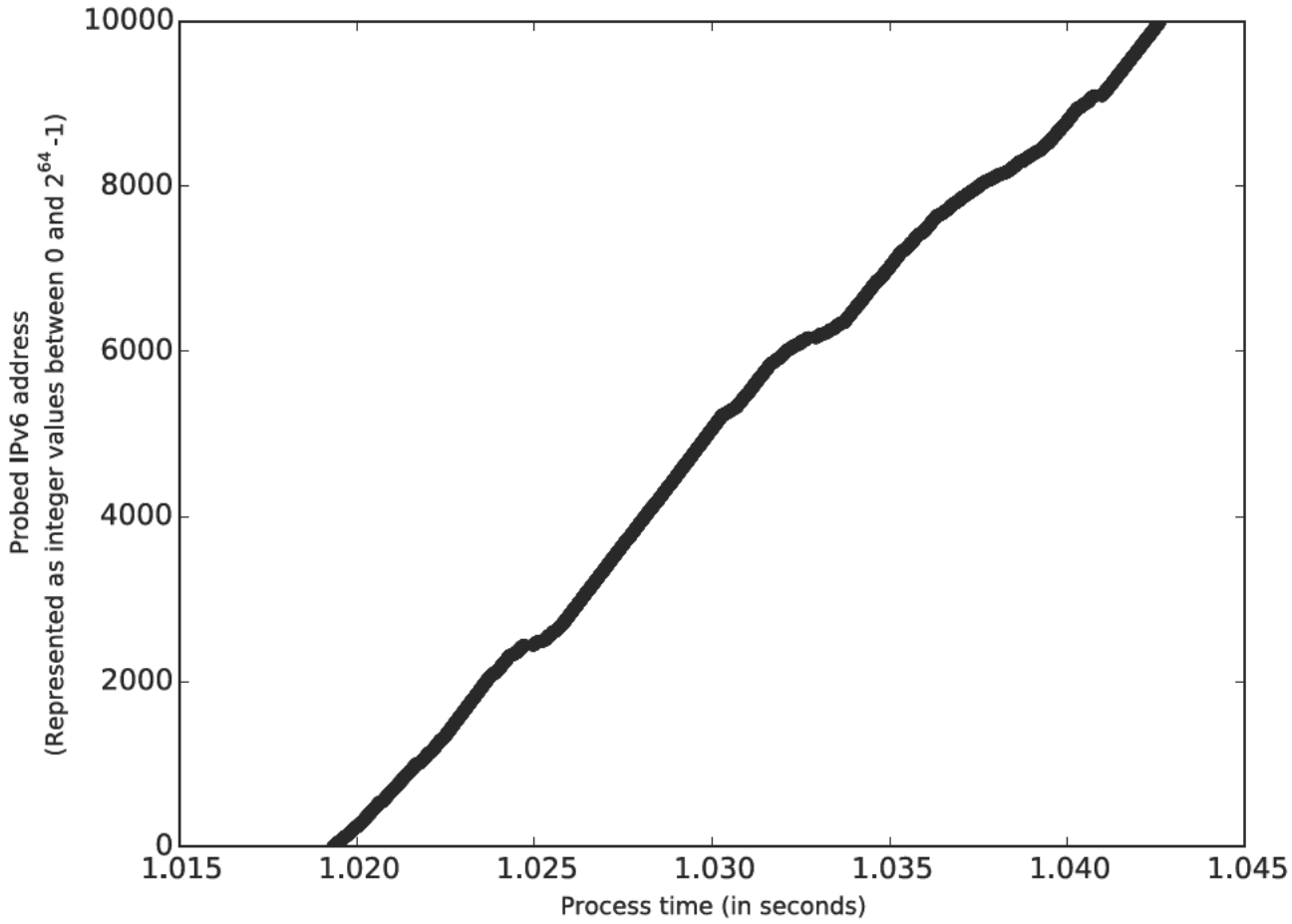


Figure 4-5: Probe distribution for the `Linear-1` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

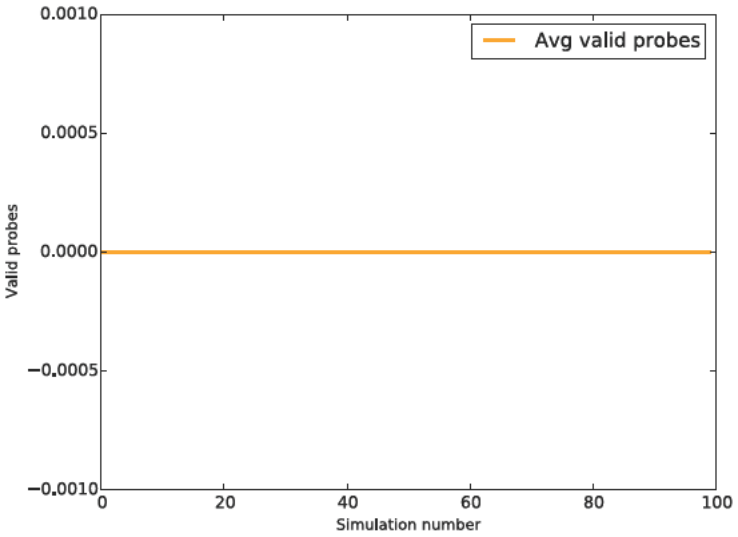
probes to 39.55% of the total valid addresses for the surveyed dataset. Since the zero-origin search performed an exhaustive search of the first 32 bits of the address space, all of the nodes that were discovered in the `Linear-1b` sub-experiment were contained in a single bucket (bucket 0), representing the first 2^{32} addresses in the address space. Overall this experiment saw a total of 16,284 unique IPv6 addresses successfully probed. These addresses were the same probed across each simulation in the sub-experiment, meaning that those 16,284 addresses were successfully probed in each of the simulations. Unfortunately the `Linear-1a` sub-experiment failed to probe a single valid host during all 100 simulations.

To confirm that the difference between the counts of the two sets were significant, the Wilcoxon Rank-Sum test was performed. The test statistic (W) from the Wilcoxon Rank-Sum test, comparing the `Linear-1a` and the sub-experiment `Linear-1b` was $W = 0.0$. The p-value for the test was $p =< 0.001$ while the target α value was $\alpha = 0.05$. The results report a significant difference between the ranked means of sub-experiment `Linear-1a` and sub-experiment `Linear-1b`. This means that for this experiment the alternative hypothesis an be accepted (i.e. that $\mu > \mu_0$). It can be concluded that there is a statistically significant difference.

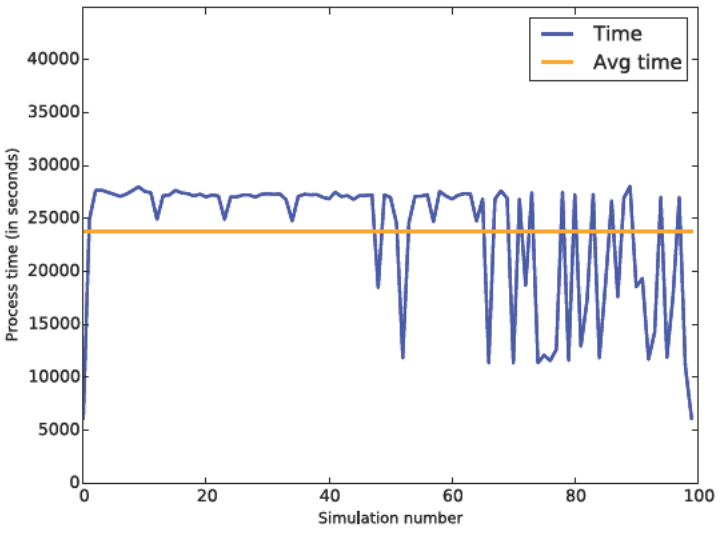
The average process completion time for the `Linear-1a` sub-experiment was 23,758 seconds, or approximately 6.5 hours (see Figure 4-6(b)). The `Linear-1b` sub-experiment saw a mean process completion time of 50,441 seconds, or approximately 14 hours (see Figure 4-7(b)).

4.3.2 Experiment Linear-2: Stochastic linear search

The stochastic `Linear-2` experiment tested the linear search algorithm against the randomised and surveyed datasets in sub-experiments `Linear-2a` and `Linear-2b` respectively. The addresses that served as the starting points for each simulation were randomly chosen during the `Linear-2` experiments. As a result, for the sub-experiments `Linear-2a` and `Linear-2b` the `linear.py` program was configured to run 100 simulations each applying the linear search algorithm to the subject dataset with the starting address for each simulation's search operation set to a randomly chosen integer between 0 and $2^{64} - 1$.

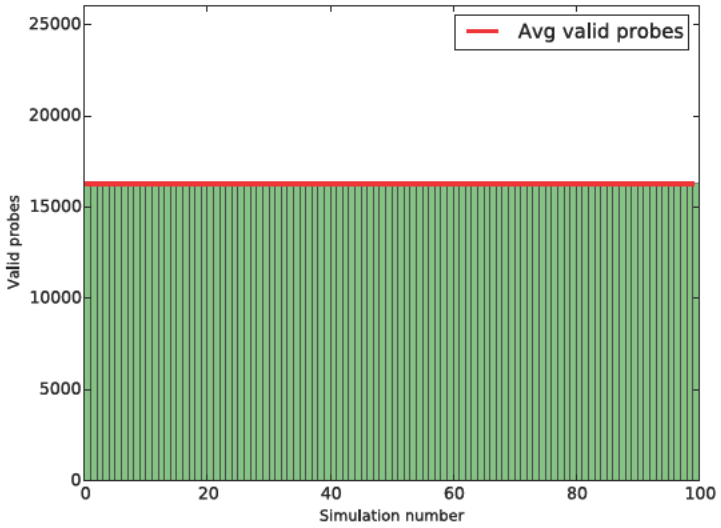


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

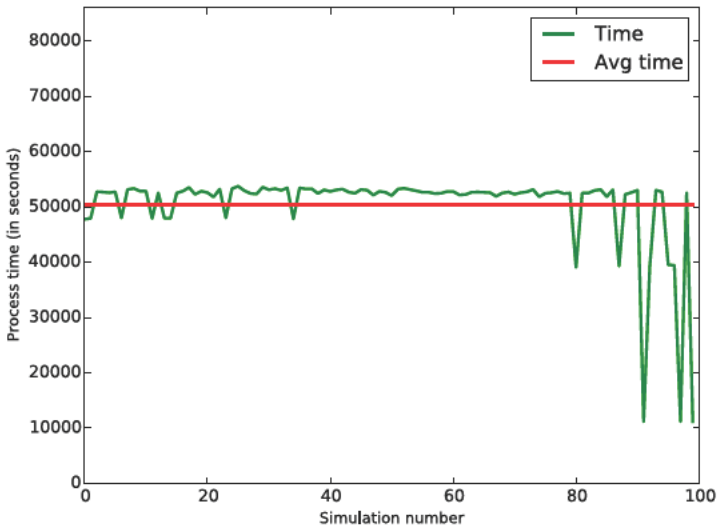


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-6: Results of the ZeroOrigin linear search against the randomised dataset (sub-experiment `Linear-1a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

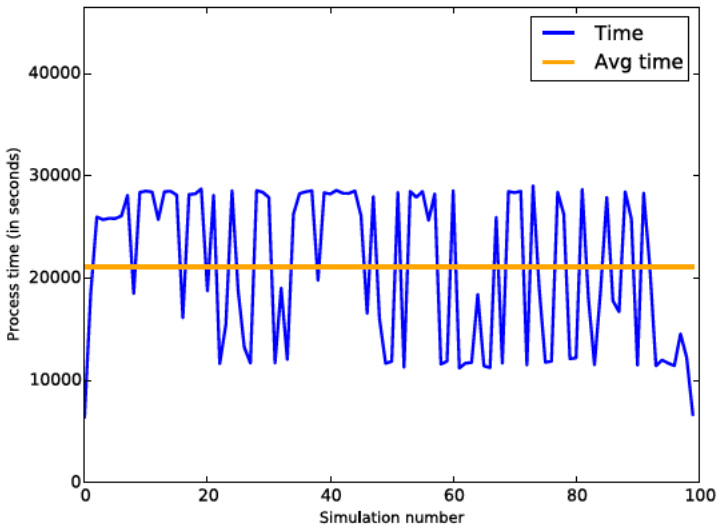
Figure 4-7: Results of the ZeroOrigin linear search against the surveyed dataset (sub-experiment `Linear-1b`) highlighting the number of valid probes delivered and process times per simulation.

A pilot study was performed prior to commencing data collection for the `Linear-2` experiment to test the linear search algorithm under experimental conditions. The pilot study for the `Linear-2` experiment tested a sample of 10,000 sequential probes across the target address space, commencing with the randomly chosen starting address of 8878645325884030976 (::7b37:48e2:0:0) and concluding at the address 8878645325884040975 (::7b37:48e2:0:270f). Figure 4-10 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the linear search algorithm probed target addresses during the `Linear-2` sub-experiments. It is evident from this graph that probing occurred in linear time as expected.

The results for the zero-origin `Linear-2a` and `Linear-2b` sub-experiments are included in Table 4.7. The `Linear-2` experiments recorded the lowest mean number of successful probes for its sub-experiments. Across all of the experiments conducted, 0 successful probes for `Linear-2a` and `Linear2b` were recorded. Since both of the sub-experiments produced counts of zero, the samples of those populations were identical, and therefore no significant difference existed between the sets (i.e. $\mu = \mu_0$).

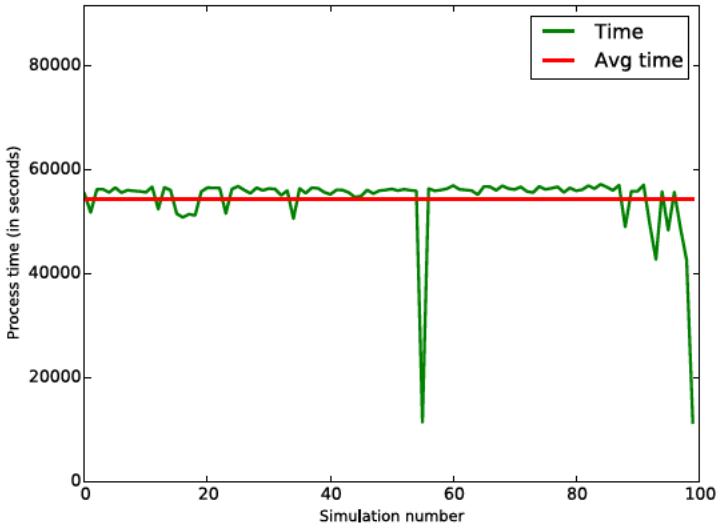
Table 4.7: The descriptive statistics of the results for the Stochastic `Linear-2` experiment simulations (experiments `Linear-2a` and `Linear-2b`).

Experimental parameters		Experiment number	
		Linear-2a	Linear-2b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	0 (0.0000%)
	Mean	0.00 (0.0000%)	0.00 (0.0000%)
	Minimum	0 (0.0000%)	0 (0.0000%)
	Median	0.0	0.0
	Total unique hosts discovered	0	0
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	29048.67	57160.77
	Mean	21119.28	54265.80
	Minimum	6440.08	11347.68



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-8: Results of the Random linear search against the randomised dataset (sub-experiment `Linear-2a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-9: Results of the Random linear search against the surveyed dataset (sub-experiment `Linear-2b`) highlighting the number of valid probes delivered and process times per simulation.

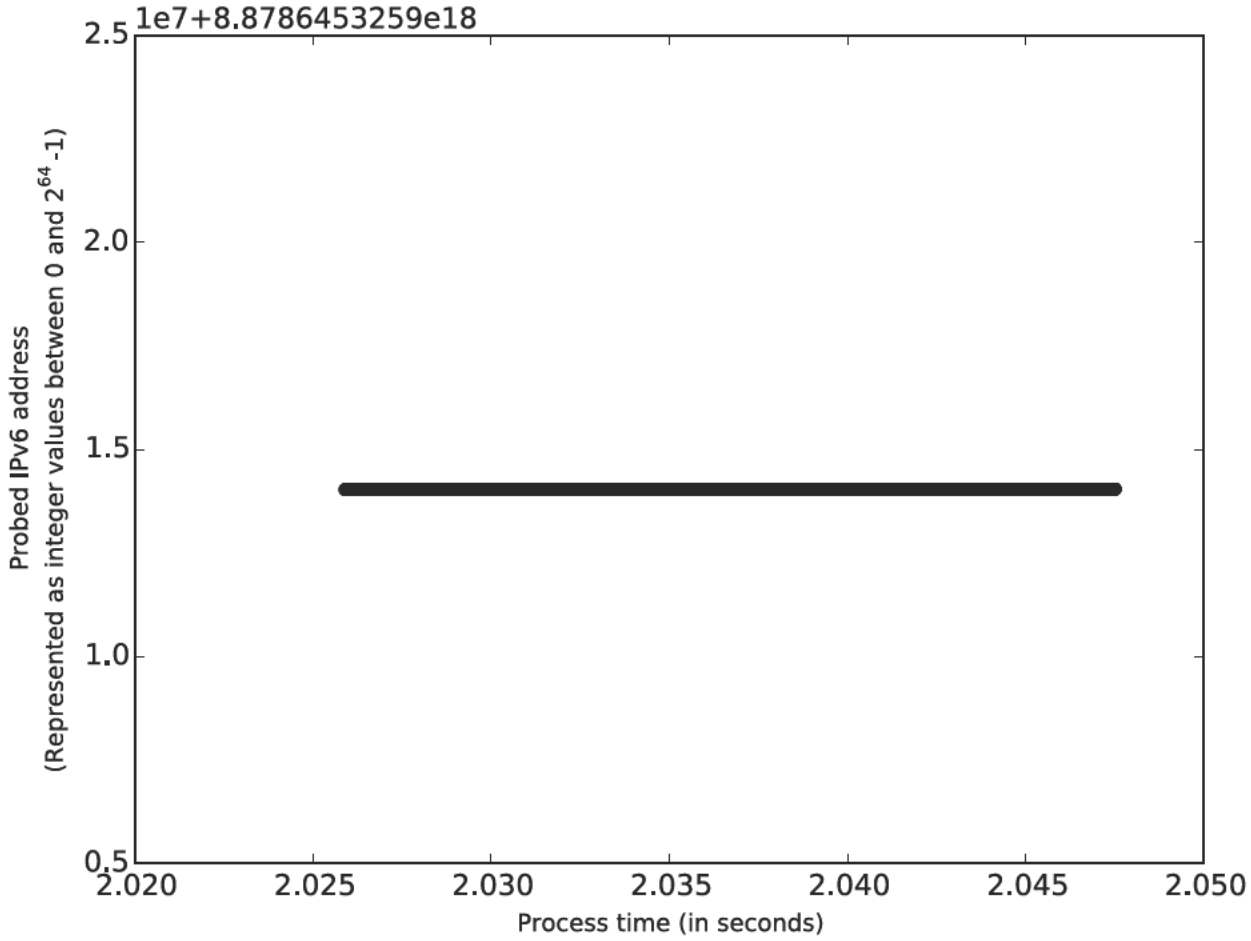


Figure 4-10: Probe distribution for the `Linear-2` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

The average process completion time for the `Linear-2` experiment was respectable and similar to that of `Linear-1`. `Linear-2a` recorded a mean process completion time of 21,119 seconds, or approximately 6 hours (see Figure 4-8(a)). The `Linear-2b` sub-experiment saw a mean process completion time of 54,266 seconds, or approximately 15 hours (see Figure 4-9(a)).

4.3.3 Experiment Linear-3: Weighted linear search

The weighted `Linear-3` experiment tested the linear search algorithm against the randomised and surveyed datasets in sub-experiments `Linear-3a` and `Linear-3b` respectively. The addresses that served as the starting points for each simulation were randomly chosen during the `Linear-3` experiments, using a weighted random choice. The resulting start address was the first address of a randomly chosen bin number. This ensured that the starting and ending addresses were on a 32 bit boundary. As a result, for the sub-experiments `Linear-3a` and `Linear-3b` the `linear.py` program was configured to run 100 simulations each applying the linear search algorithm to the subject dataset with the starting address for each simulation's search operation set to a randomly chosen bin number (b) between 0 and $2^{32}-1$, such that the start address ($start$) was calculated to be:

$$start = b * 2^{32} \tag{4.2}$$

A pilot study was performed prior to commencing data collection for the `Linear-3` experiment to test the linear search algorithm under experimental conditions. The pilot study for the `Linear-3` experiment tested a sample of 10,000 sequential probes across the target address space, commencing with the randomly chosen starting address of 281560876056576 (::1:14:0:0) and concluding at the address 281560876066575 (::1:14:0:270f). Figure 4-11 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the linear search algorithm probed target addresses during the `Linear-3` sub-experiments. Similarly to the `Linear-1` and `Linear-2` experiments, it is evident from this graph that probing occurred in linear time as expected.

Each simulation probed 2^{32} sequential addresses from a starting point selected

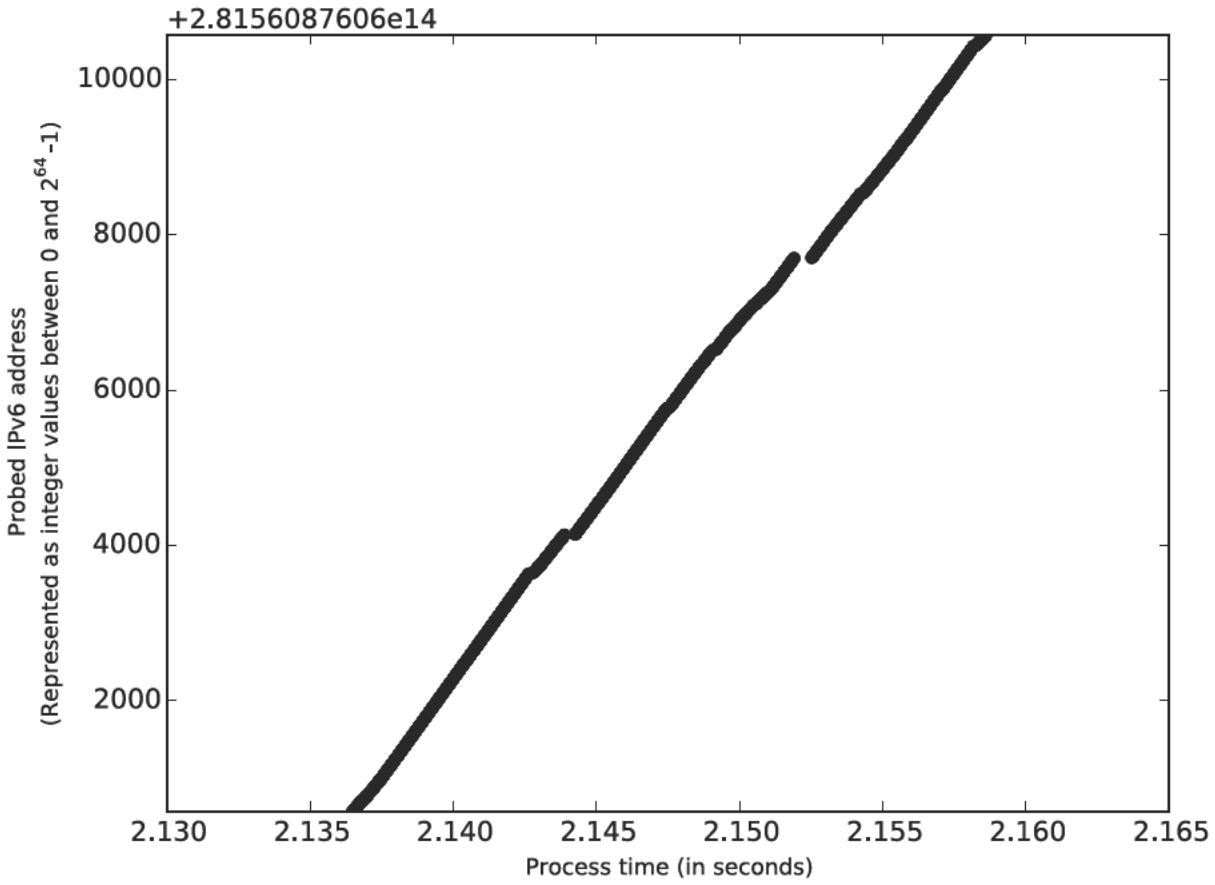


Figure 4-11: Probe distribution for the `Linear-3` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

through the means described above. The results for the zero-origin `Linear-3a` and `Linear-3b` sub-experiments are included in Table 4.8. The `Linear-3` experiments recorded the lowest mean number of successful probes for its sub-experiments. Across all of the experiments conducted a mean of 0.49 and 8642.1 successful probes for `Linear-3a` (see Figure 4-12(a)) and `Linear-3b` (see Figure 4-13(a)) were observed respectively.

The results indicate that across the 100 simulations conducted for the `Linear-3a` sub-experiment, a maximum of one single successful probe was recorded. The minimum number of successful probes recorded was 0, revealing an average of 0.49 successful probes per simulation. It was also observed that 49 unique nodes were successfully probed across all 100 simulations. There were no common nodes probed throughout every simulation.

Figure 4-14 lists the unique IPv6 addresses that were successfully probed throughout the 100 simulations, chunked into 2^{32} buckets with the top 25 observed buckets displayed of 2^{32} addresses. Only the top 25 buckets are included in Figure 4-14. The frequency distribution displays an even distribution of valid probes across the address space, indicating that at most, only a single address was detected in a bucket during the `Linear-3a` sub-experiment.

By contrast, the results for the `Linear-3b` sub-experiment indicate that across the 100 simulations conducted, a maximum number of 16,284 successful probes observed in a single simulation was observed. The minimum number of successful probes observed across the simulations was 0. These results represent the third highest average number of successful probes recorded for a sub-experiment in this research. It was observed that 26,796 unique nodes were successfully probed across all 100 simulations. Figure 4-15 displays the unique IPv6 addresses that were successfully probed throughout the 100 simulations, chunked into 2^{32} buckets with the top 25 observed buckets displayed of 2^{32} addresses. The top 25 observed buckets are shown. The frequency distribution displays that the majority of valid probes detected in the `Linear-3b` sub-experiment were contained to the first bucket.

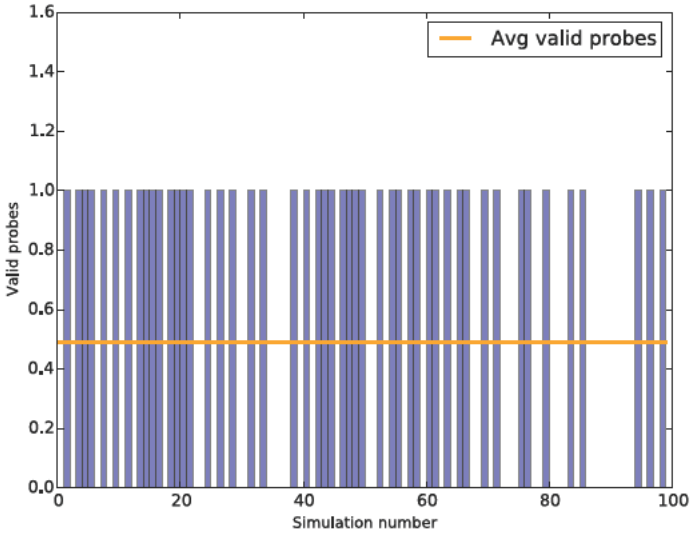
The results of `Linear-3a` and `Linear-3b` were compared to determine if the difference between the populations were significant. In particular whether $\mu > \mu_0$. The

Table 4.8: The descriptive statistics of the results for the Weighted `Linear-3` experiment simulations (experiments `Linear-3a` and `Linear-3b`).

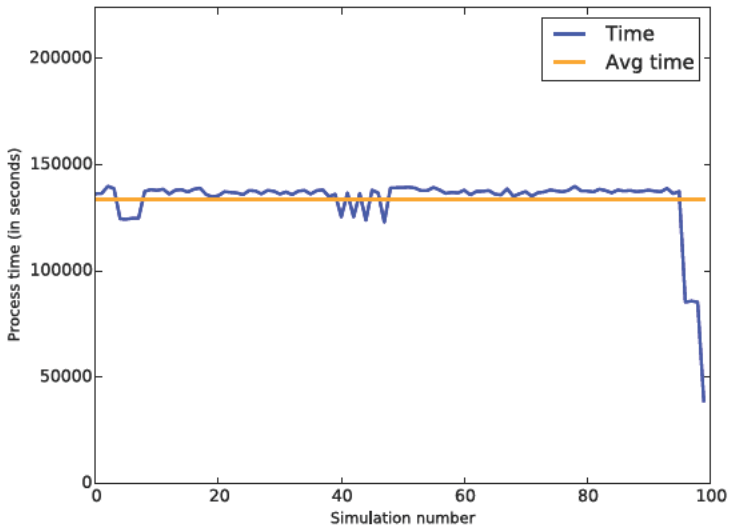
Experimental parameters		Experiment number	
		Linear-3a	Linear-3b
Valid probes		(/50000)	(/41171)
	Maximum	1 (0.0020%)	16284 (39.5521%)
	Mean	0.49 (0.0010%)	8642.09 (20.9907%)
	Minimum	0 (0.0000%)	0 (0.0000%)
	Median	0.0	16284.0
	Total unique hosts discovered	49	26796
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	139838.82	139353.21
	Mean	133798.51	132887.30
	Minimum	38883.56	38336.12

difference between the ranked means of sub-experiment `Linear-3a` and sub-experiment `Linear-3b` was significant with a test statistic (W) of $W = 170.0$, and the p-value of $p =< 0.001$. This highlights a significant difference between the two groups.

The average process completion time for the `Linear-3` was substantially greater than the other two experiments involving the linear search algorithm and `linear.py`. `Linear-3a` recorded a mean process completion time of 133,799 seconds, or approximately 37 hours (see Figure 4-12(b)). The `Linear-3b` sub-experiment saw a mean process completion time of 132,887 seconds, or approximately 37 hours (see Figure 4-13(b)). The processing times for this experiment were amongst the longest recorded times during the research.

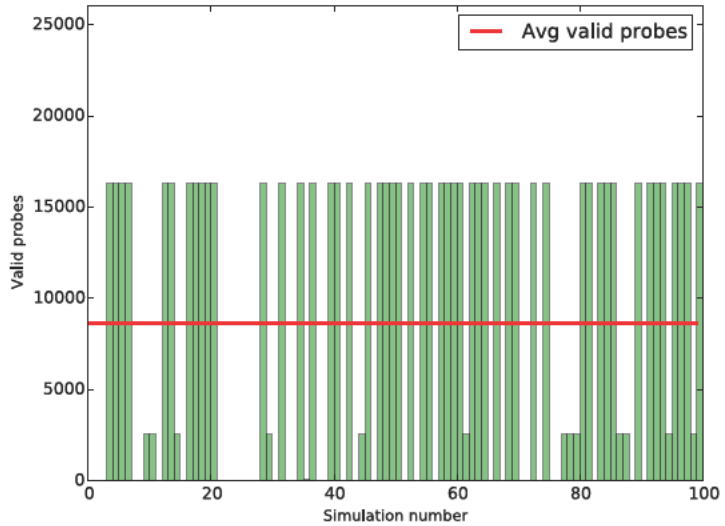


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

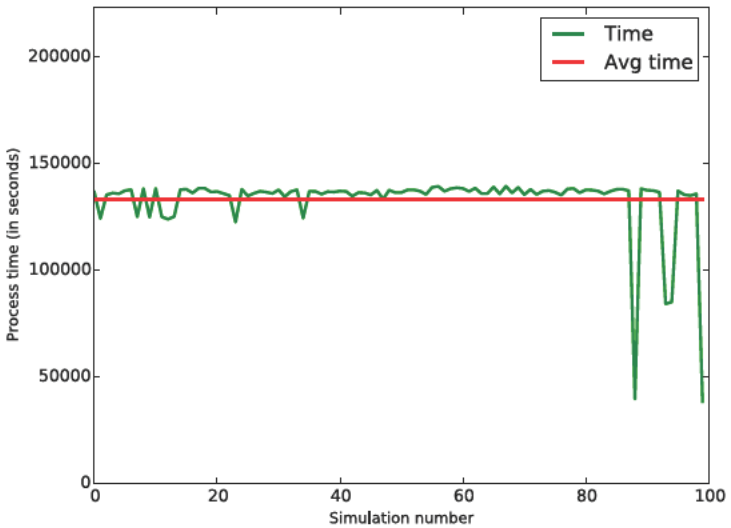


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-12: Results of the Weighted linear search against the randomised dataset (sub-experiment `Linear-3a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-13: Results of the Weighted linear search against the surveyed dataset (sub-experiment `Linear-3b`) highlighting the number of valid probes delivered and process times per simulation.

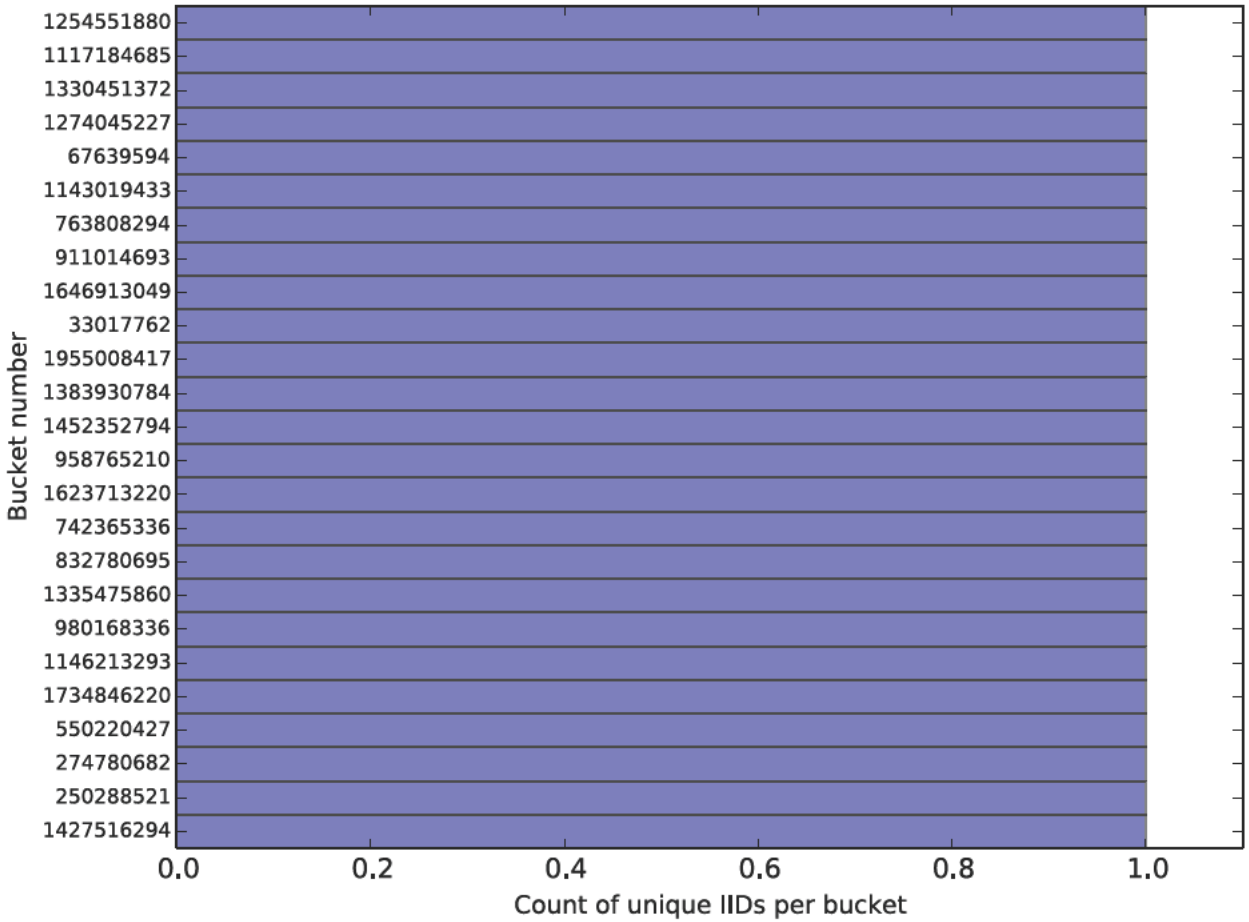


Figure 4-14: The frequency of unique identified nodes across all simulations in the `Linear-3a` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

‘

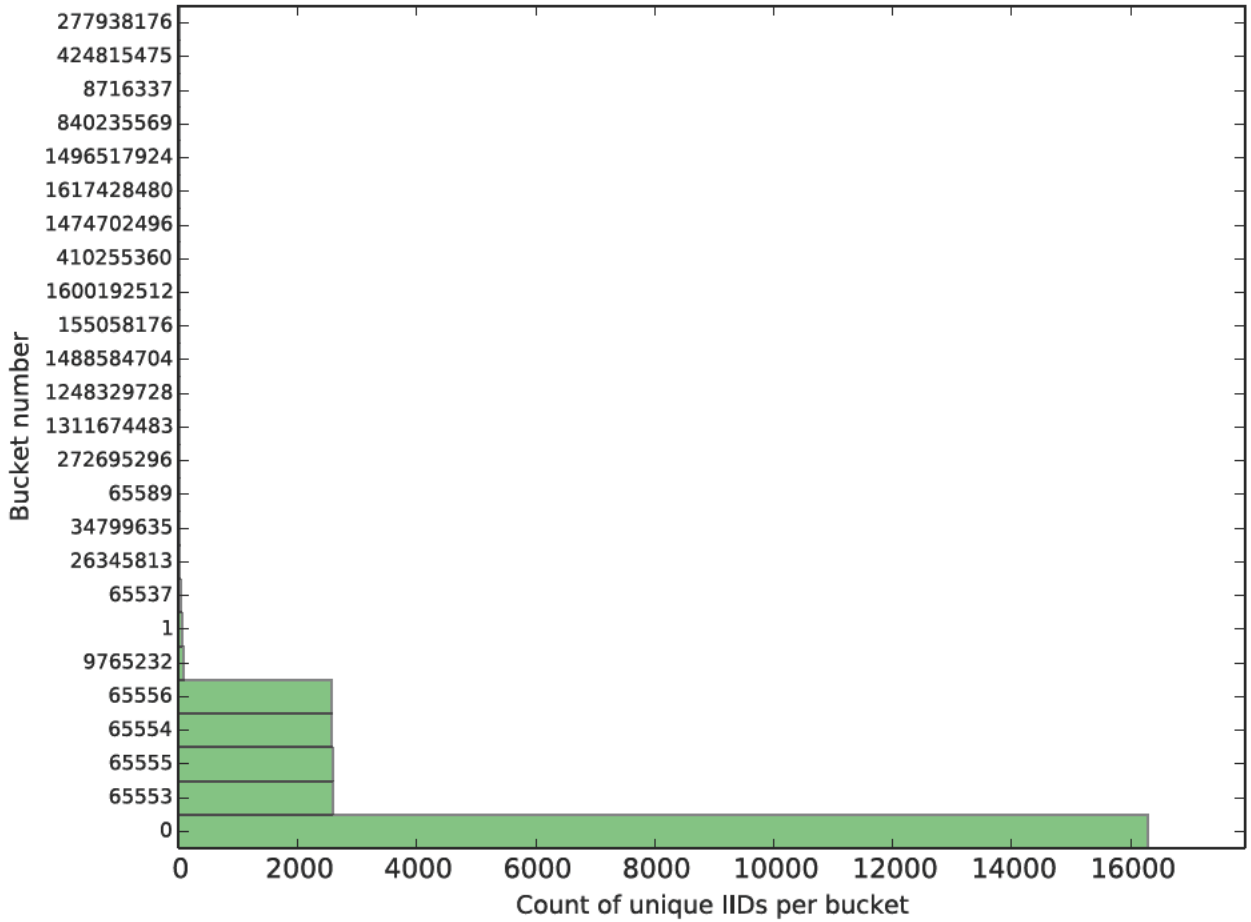


Figure 4-15: The frequency of unique identified nodes across all simulations in the `Linear-3b` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

4.4 Stripe search

The stripe search algorithm was designed as an extension of the conventional linear search algorithm. Rather than perform a conventional, exhaustive sequential search (which is conventional with a linear search), the stripe search performed sequential searches of chunks of the entire address space. The algorithm was the subject of two main experiments; the `Stripe-1`, which saw the algorithm commence search operations at randomly generated starting points within the first chunk of the address space; and the `Stripe-2` experiment, which tested the algorithm with the starting point of the sequential search at the first address of each chunk in the address space (i.e. address `::`).

The Python program `stripe.py` was used to implement the stripe search algorithm. The `stripe.py` program was configured to deliver a maximum of 2^{32} probes, per simulation, to IPv6 addresses from a designated starting address that varied per experiment to align with the specifications detailed above. The program was also configured to perform 100 simulations for each sub-experiment. The outcomes of these experiments are detailed in the following sections.

4.4.1 Experiment Stripe-1: Stochastic stripe search

The stochastic `Stripe-1` experiment tested the stripe search algorithm against the randomised and surveyed datasets in sub-experiments `Stripe-1a` and `Stripe-1b` respectively. The addresses that served as the starting points for each simulation were randomlly chosen during the `Stripe-1` experiments, using a weighted random choice. The resulting start address was the first address of a randomly chosen bin number. This ensured that the starting and ending addresses were on a 32 bit boundary. As a result, for the sub-experiments `Stripe-1a` and `Stripe-1b` the `stripe.py` program was configured to run 100 simulations each applying the stripe search algorithm to the subject dataset with the starting address for each simulation's search operation set to a randomly chosen bin number between 0 and $2^{16} - 1$.

A pilot study was performed prior to commencing data collection for the `Stripe-1` experiment. The pilot study for the `Stripe-1` experiment tested a sample of 10,000

sequential probes across the target address space, commencing with the randomly chosen starting address of 74174821276193 (::4376:2bdf:9a21) and concluding at the address 2607449611673090 (::9:4376:2bdf:9e02). Figure 4-19 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how the stripe search algorithm probed target addresses during the `Stripe-1` sub-experiments. The striping nature of the algorithm is evident in Figure 4-19.

Each simulation probed 2^{32} addresses from a starting point selected through the means described above. The results for the stochastic `Stripe-1a` and `Stripe-1b` sub-experiments are included in Table 4.9. The `Stripe-1` experiments recorded the lowest mean number of successful probes for its sub-experiments. Across all of the experiments conducted a mean of 0 successful probes for `Stripe-1a` (see Figure 4-17(a)) and `Stripe-1b` (see Figure 4-18(a)) were observed. This indicates that unfortunately the `Stripe-1` experiment failed to probe a single valid host. Since both of the sub-experiments revealed counts of zero, the samples of the population were identical, and therefore no significant difference existed between the sets (i.e. $\mu = \mu_0$).

Table 4.9: The descriptive statistics of the results for the Stochastic `Stripe-1` experiment simulations (experiments `Stripe-1a` and `Stripe-1b`).

Experimental parameters		Experiment number	
		Stripe-1a	Stripe-1b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	0 (0.0000%)
	Mean	0.00 (0.0000%)	0.00 (0.0000%)
	Minimum	0 (0.0000%)	0 (0.0000%)
	Median	0.0	0.0
	Total unique hosts discovered	0	0
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	30474.32	51639.36
	Mean	25543.13	49003.80
	Minimum	5322.36	10781.34

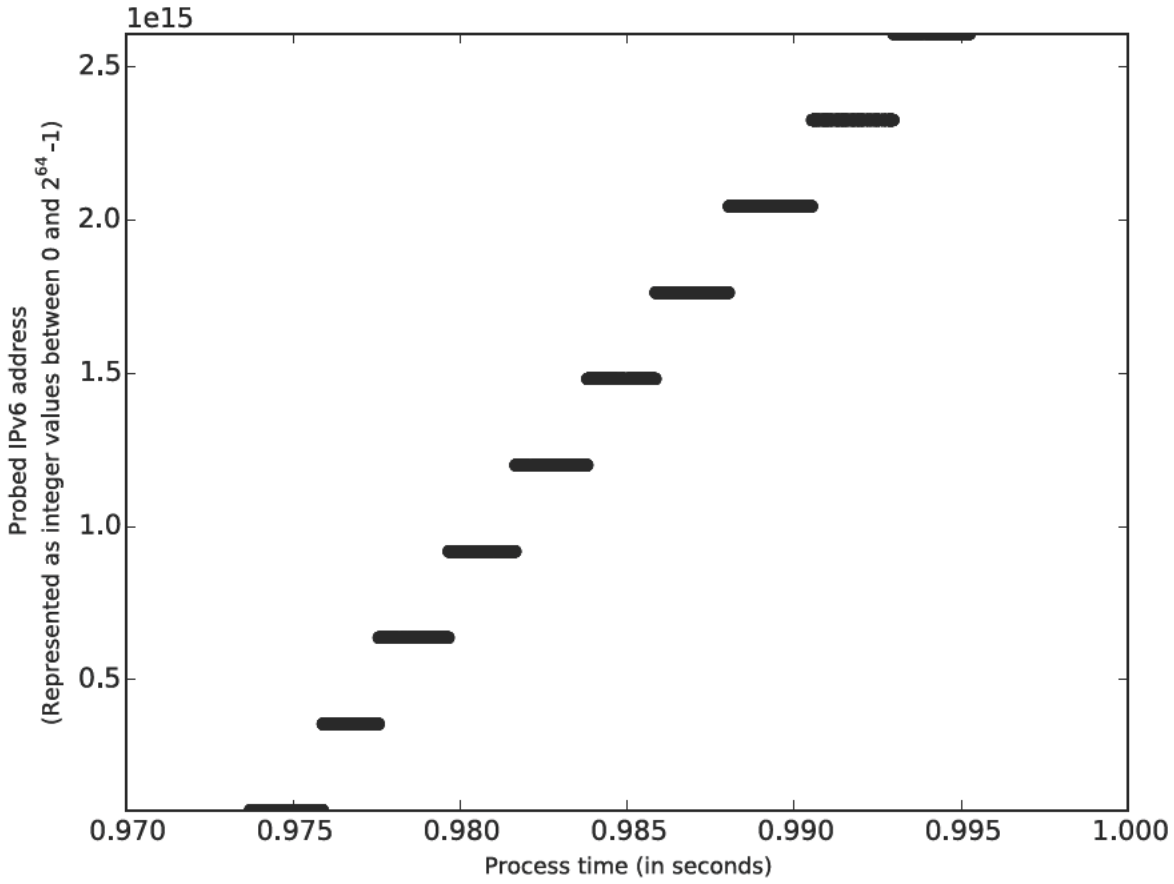


Figure 4-16: Probe distribution for the `Stripe-1` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

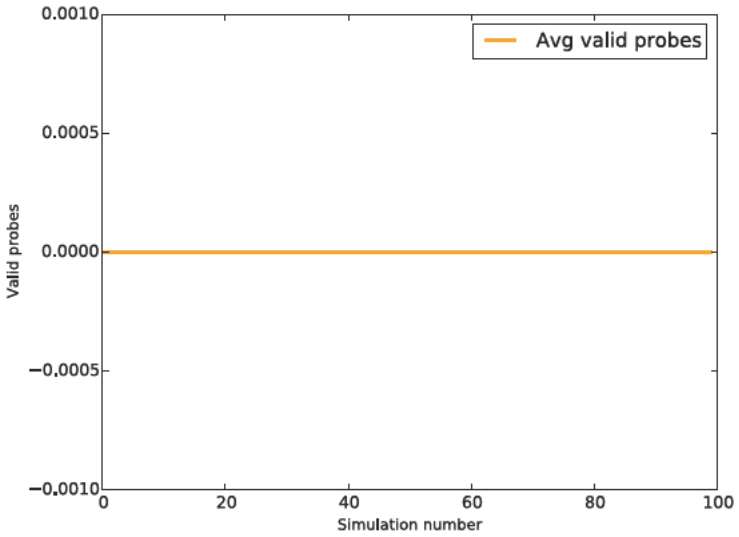
The average process completion time for the `Stripe-1` was similar to that of the `Linear-1` and `Linear-2` experiments. `Stripe-1a` recorded a mean process completion time of 25,543 seconds, or approximately 7 hours (see Figure 4-17(b)). The `Stripe-1b` sub-experiment recorded a mean process completion time of 49,004 seconds, or approximately 13.5 hours (see Figure 4-18(b)).

4.4.2 Experiment Stripe-2: Zero-origin stripe search

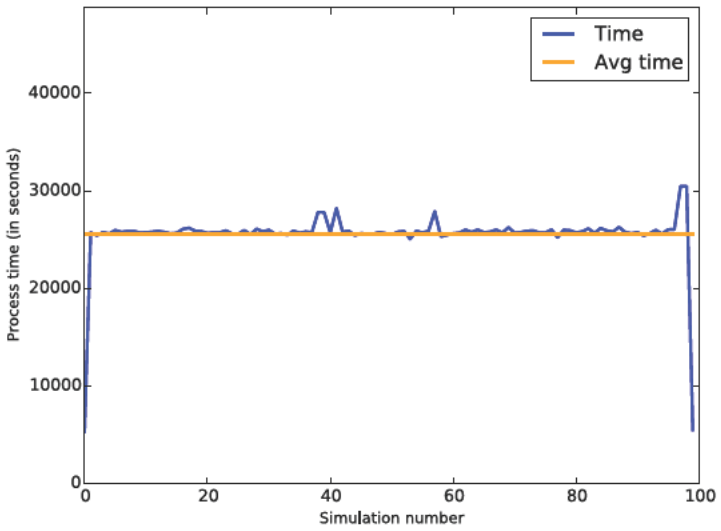
The zero-origin `Stripe-2` experiment tested the stripe search algorithm against the randomised and surveyed datasets in sub-experiments `Stripe-2a` and `Stripe-2b` respectively. The addresses that served as the starting points for the search operations were set to the first address in the address space (`::`) for `Stripe-2` experiments. Therefore, for the sub-experiments `Stripe-2a` and `Stripe-2b` the `stripe.py` program was configured to run 100 simulations each applying the stripe search algorithm to the subject dataset with the starting address for each simulation's search operation set to 0. Similar to `Linear-1` this experiment was controlled and deterministic due to the start address that held constant across all simulations.

A pilot study was performed prior to commencing data collection for the `Stripe-2` experiment. The pilot study for the `Stripe-2` experiment tested a sample of 10,000 sequential probes across the target address space, commencing with the randomly chosen starting address of 0 (::) and concluding at the address (::9:0:0:3e4). Figure 4-19 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the stripe search algorithm probed target addresses during the `Stripe-2` sub-experiments. As with the pilot study for the `Stripe-1` experiment, the striping nature of the algorithm can be seen in Figure 4-19.

As mentioned previously, 100 simulations were conducted for each of the `Stripe-2a` and `Stripe-2b` sub-experiments. Each simulation attempted to search the address space using 4,294,967,296 probes spread across 2^{16} buckets (each bucket was 2^{48} addresses wide). Although `Stripe-2a` did not record a single probe against a valid host, `Stripe-2b` recorded one of the highest average successful probe counts across all of the experiments conducted. During `Stripe-2b` an average of 4,150 successful probes per simulation were recorded (Table 4.10). It was observed that a total of 4,150 unique nodes were probed across all 100 simulations. These 4,150 unique nodes were probed in every one of the 100 simulations for this sub-experiment. The breakdown of the frequency of unique IPv6 addresses detected per bucket across all simulations is included in Figure 4-20. The frequency distribution displays that the majority of valid probes

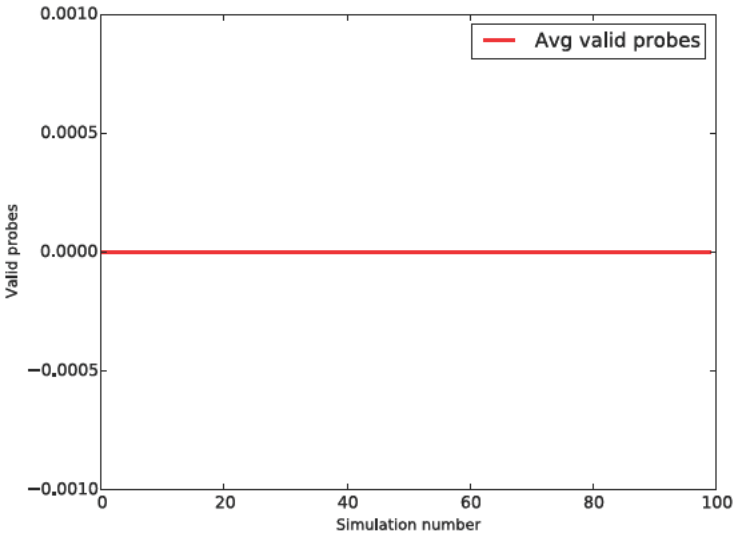


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

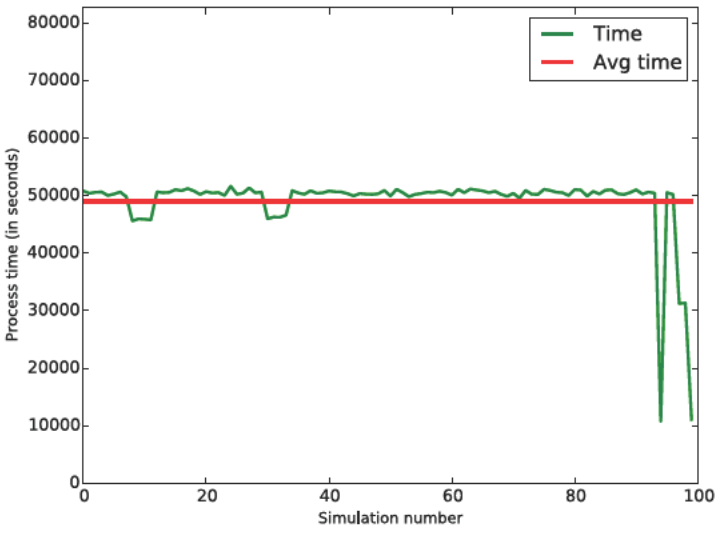


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-17: Results of the Random stripe search against the randomised dataset (sub-experiment `Stripe-1a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-18: Results of the Random stripe search against the surveyed dataset (sub-experiment `Stripe-1b`) highlighting the number of valid probes delivered and process times per simulation.

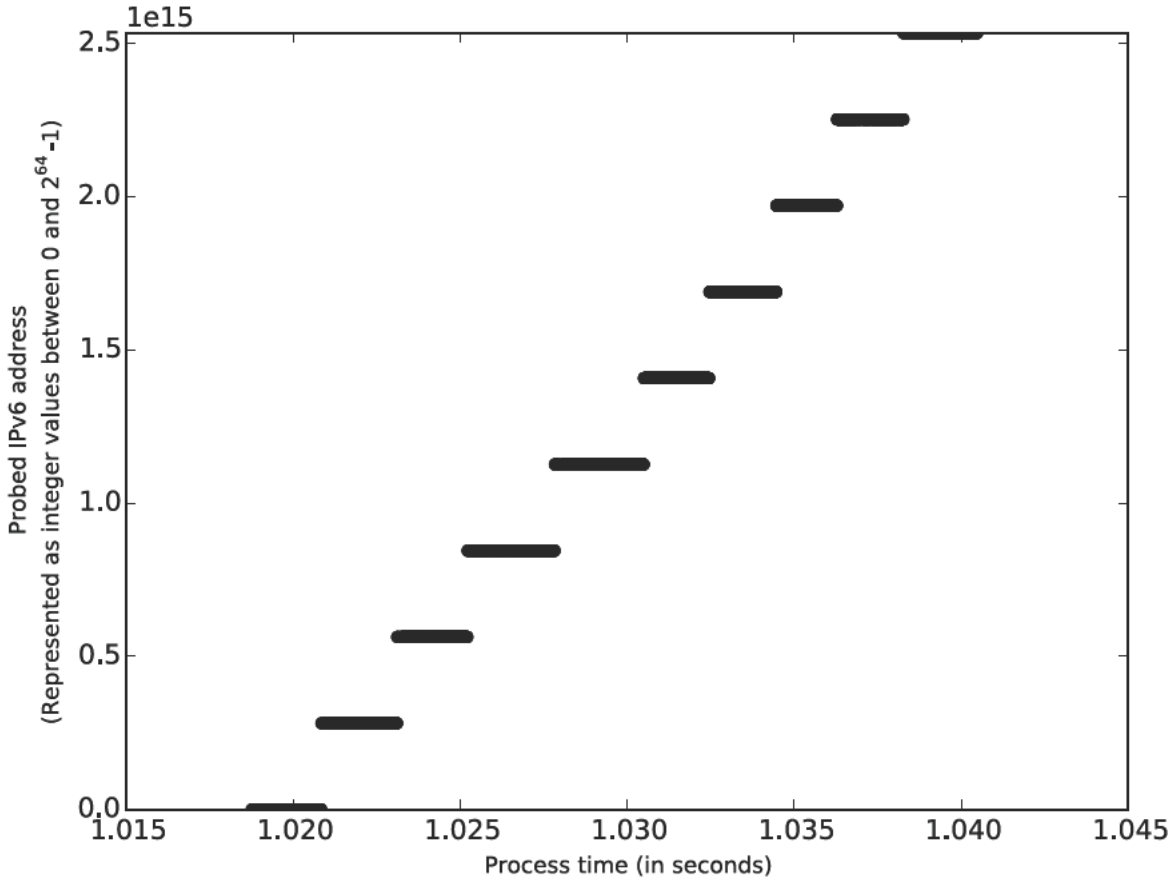


Figure 4-19: Probe distribution for the `Stripe-2` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

detected in the `Stripe-2b` sub-experiment were contained to the first bucket.

The populations for the `Stripe-2a` and `Stripe-2b` sub-experiments were compared using a Wilcoxon Rank-Sum test to determine whether they differed significantly. The results highlighted a significant difference between the two data sets at the 95% confidence interval. The test statistic was $W = 0.0$, and the p-value was $p =< 0.001$ with a target α value of $\alpha = 0.05$.

Table 4.10: The descriptive statistics of the results for the Zero Origin `Stripe-2` experiment simulations (experiments `Stripe-2a` and `Stripe-2b`).

Experimental parameters		Experiment number	
		Stripe-2a	Stripe-2b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	4150 (10.0799%)
	Mean	0.00 (0.0000%)	4150.00 (10.0799%)
	Minimum	0 (0.0000%)	4150 (10.0799%)
	Median	0.0	4150.0
	Total unique hosts discovered	0	4150
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	45147.45	52169.45
	Mean	27753.18	50041.92
	Minimum	5316.17	10721.14

The average process completion times for the `Stripe-2` experiment was similar to that of the `Stripe-1` experiment. `Stripe-2a` recorded a mean process completion time of 27,753 seconds, or approximately 8 hours (see Figure 4-21(b)). The `Stripe-2b` sub-experiment recorded a mean process completion time of 50,420 seconds, or approximately 14 hours (see Figure 4-22(b)).

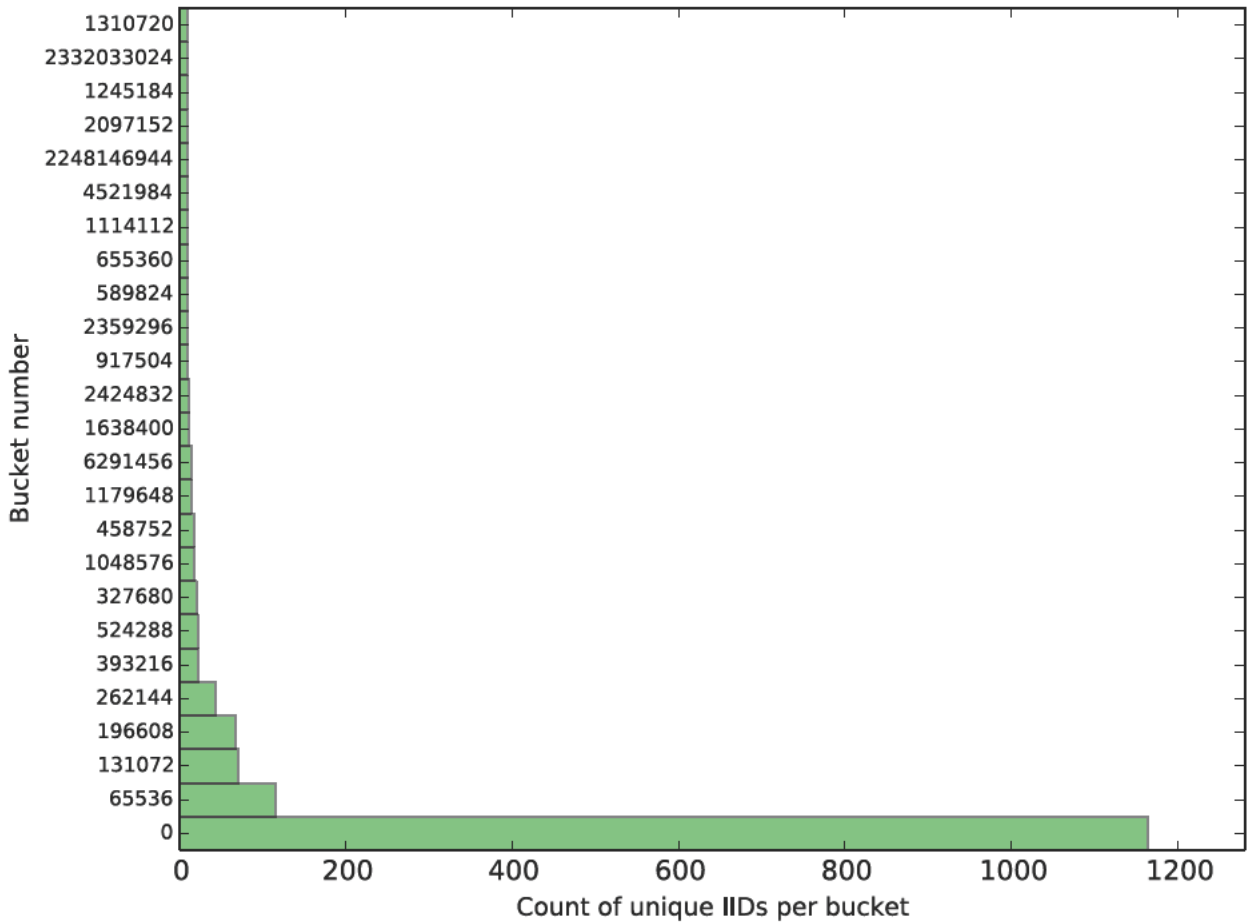
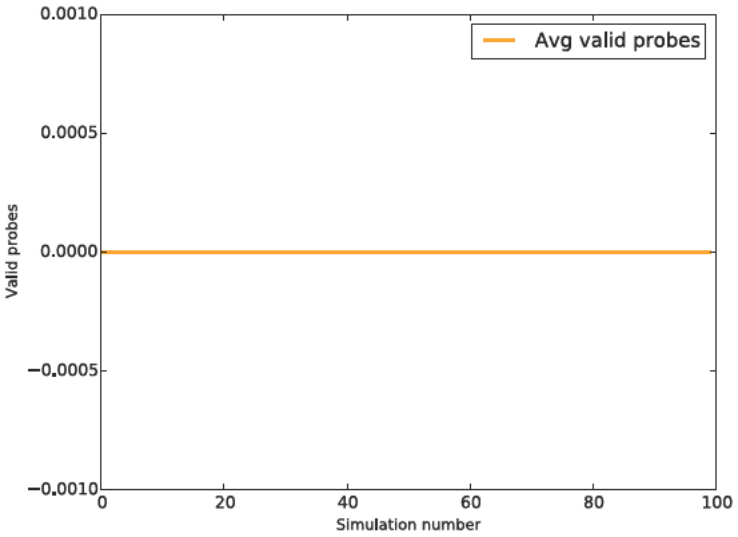
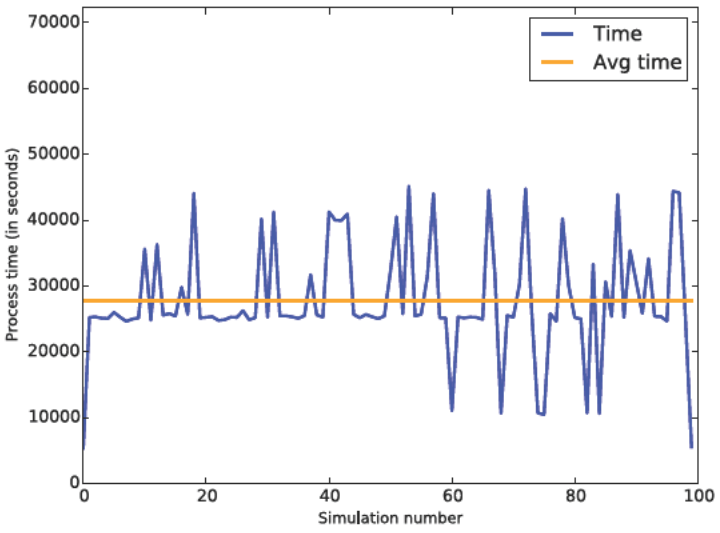


Figure 4-20: The frequency of unique identified nodes across all simulations in the `Stripe-2b` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

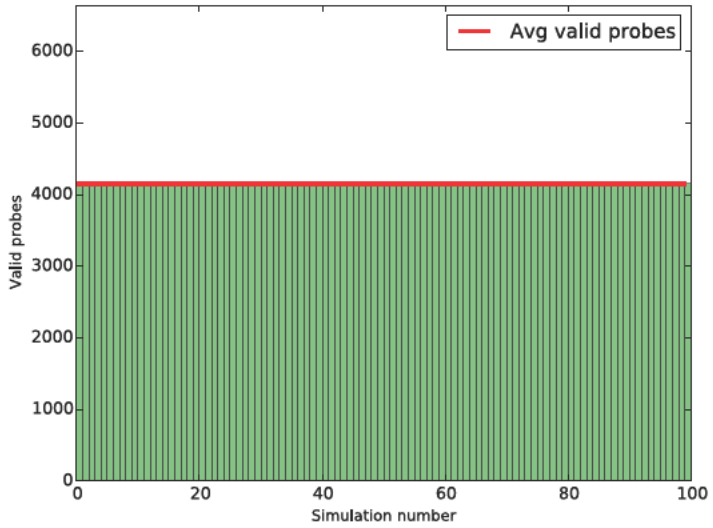


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

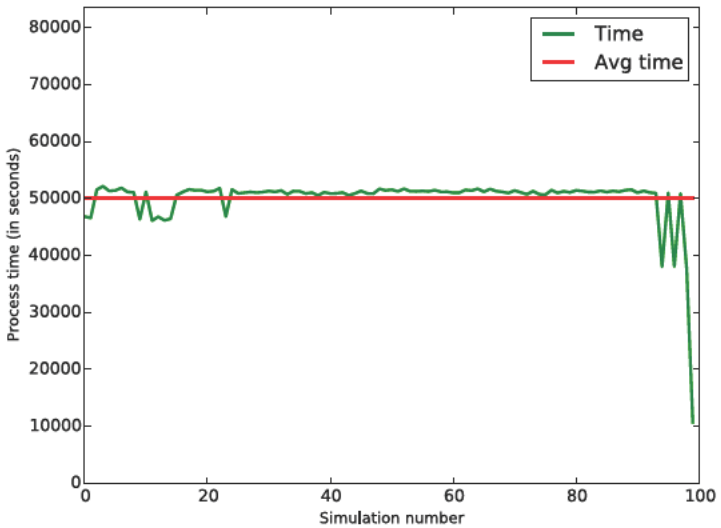


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-21: Results of the ZeroOrigin stripe search against the randomised dataset (sub-experiment `Stripe-2a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-22: Results of the ZeroOrigin stripe search against the surveyed dataset (sub-experiment `Stripe-2b`) highlighting the number of valid probes delivered and process times per simulation.

4.5 Monte Carlo search

The Monte Carlo stochastic search algorithm was designed to test a uniformly distributed randoml search of the 64 bit IPv6 network address space. This technique is commonly employed in search algorithms that are used to enumerated IPv4 networks, however has not been applied to IPv6. The algorithm was the subject of two main experiments; the `MonteCarlo-1`, which saw the algorithm search randomlly chosen addresses without influence; and the `MonteCarlo-2` which influenced the selection process by binning the address space, and then weighting the bins using a weighted random choice.

The Python program `monte_carlo.py` was used to implement the Monte Carlo stochastic search algorithm. The `monte_carlo.py` program was configured to deliver a maximum of 2^{32} probes, per simulation, to IPv6 addresses aligning with the specifications detailed above. The program was also configured to perform 100 simulations for each sub-experiment. The outcomes of these experiments are detailed below.

4.5.1 Experiment MonteCarlo-1: Stochastic Monte Carlo search

The stochastic `MonteCarlo-1` experiment tested the Monte Carlo search algorithm against the randomised and surveyed datasets in sub-experiments `MonteCarlo-1a` and `MonteCarlo-1b` respectively. For this experiment `monte_carlo.py` was configured to select target addresses through the use of a pseudorandom process. A seeded Mersenne Twister-based PRNG was employed to generate addresses between 0 and $2^{64} - 1$. This PRNG was seeded using bytes from the experimental computer's kernel random-generator (i.e. `/dev/urandom`).

A pilot study was performed for the `MonteCarlo-1` experiment prior to commencing data collection. The pilot study for the `MonteCarlo-1` experiment tested a sample of 10,000 randomlly chosen probes across the target address space. Figure 4-23 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the Monte Carlo search algorithm probed target addresses during the `MonteCarlo-1` sub-experiments. It is evident from Figure 4-23 that there are patterns in the distribution

of probes across the address space - it is not indeterminably random, and is certainly not cryptographically secure. For the purposes of this experiment these traits have little impact since the aim of the experiment was to achieve a uniform distribution of probes across the address space for which the pilot study demonstrates.

Each simulation probed 2^{32} randomly selected addresses from a starting point selected through the means described above. The results for the stochastic `MonteCarlo-1a` and `MonteCarlo-1b` sub-experiments are included in Table 4.11. The `MonteCarlo-1` experiments recorded the lowest mean number of successful probes for its sub-experiments. Across all of the experiments conducted a mean of 0 successful probes for `MonteCarlo-1a` and `MonteCarlo-1b` were observed. This indicates that similar to the `Linear-2` and `Stripe-1` experiments, unfortunately the `MonteCarlo-1` experiment failed to probe a single valid host. Since both of the sub-experiments revealed counts of zero, the populations were identical, and therefore no significant difference existed between the samples (i.e. $\mu = \mu_0$).

Table 4.11: The descriptive statistics of the results gathered from Stochastic Monte Carlo simulations (experiments `MonteCarlo-1a` and `MonteCarlo-1b`).

Experimental parameters		Experiment number	
		MonteCarlo-1a	MonteCarlo-1b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	0 (0.0000%)
	Mean	0.00 (0.0000%)	0.00 (0.0000%)
	Minimum	0 (0.0000%)	0 (0.0000%)
	Median	0.0	0.0
	Total unique hosts discovered	0	0
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	219874.71	215418.31
	Mean	175437.93	175111.66
	Minimum	71295.43	71284.27

The average process completion times for the `MonteCarlo-1` experiment were amongst the highest recorded. `MonteCarlo-1a` recorded a mean process completion time of

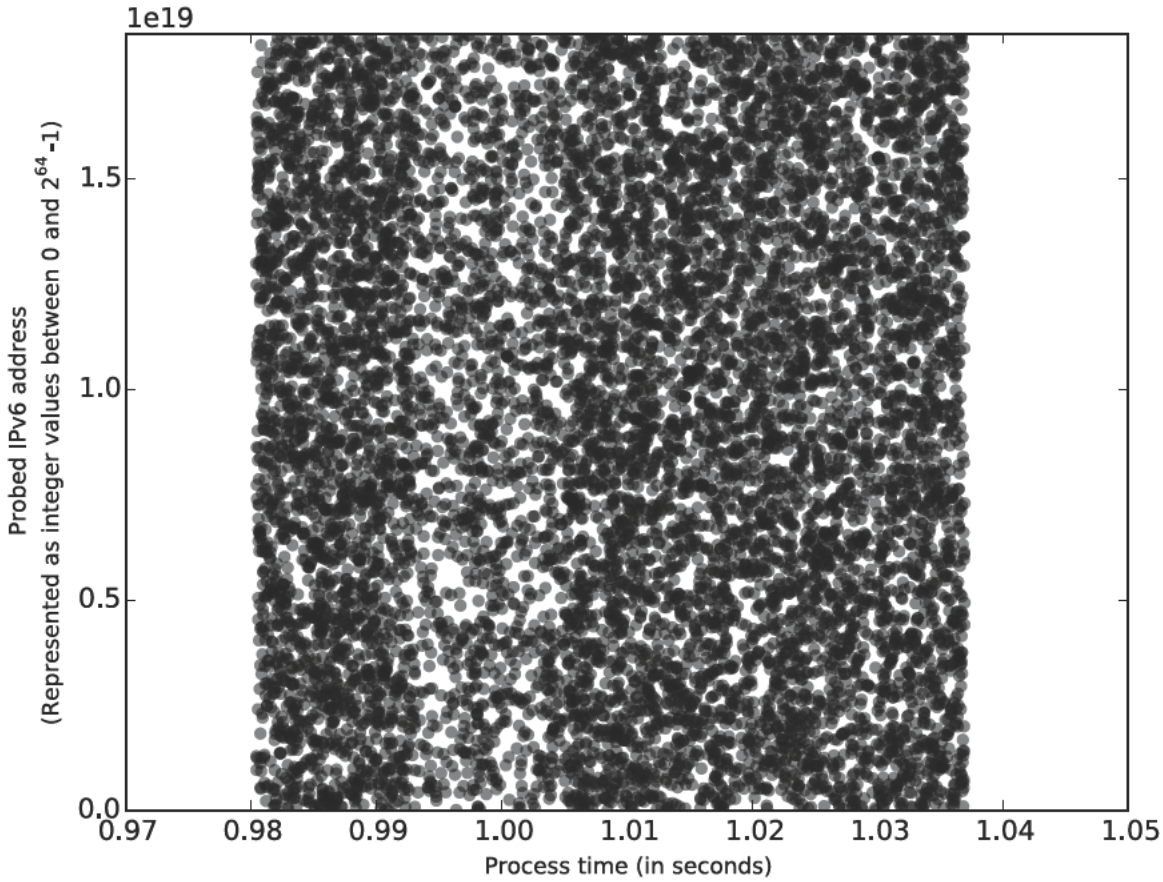


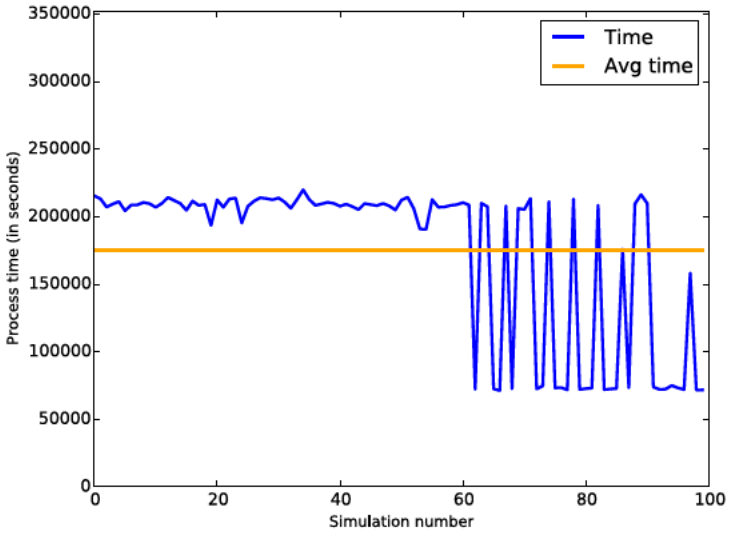
Figure 4-23: Probe distribution for the `MonteCarlo-1` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

175,438 seconds, or approximately 49 hours (see Figure 4-24(a)). The `MonteCarlo-1b` sub-experiment recorded a similar mean process completion time of 175,112 seconds, or approximately 49 hours (see Figure 4-25(a)). The fluctuations that can be seen in Figure 4-25(a) are likely due to a simulation artefact whereby the computer cluster that was used for the experimentation was over subscribed with other tasks. Unfortunately due to the research computer cluster being a communal resource, isolated access could not be controlled throughout the entire experimentation process. Consequently some of the experiments' process times were affected by the increased load.

4.5.2 Experiment MonteCarlo-2: Weighted Monte Carlo search

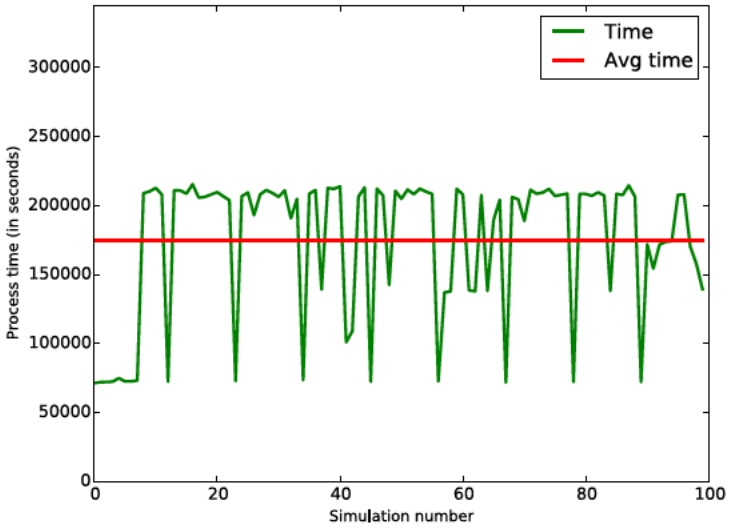
The weighted `MonteCarlo-2` experiment tested the Monte Carlo search algorithm against the randomised and surveyed datasets in sub-experiments `MonteCarlo-2a` and `MonteCarlo-2b` respectively. For this experiment `monte_carlo.py` was configured to select target addresses through the use of a weighted-pseudorandom choice process. The same seeded Mersenne Twister-based PRNG was employed to generate addresses between 0 and $2^{64} - 1$ was employed for this experiment as with `MonteCarlo-2`. This PRNG was also seeded using bytes from the experimental computer's kernel random-generator (i.e. `/dev/urandom`). For each probe delivered, the PRNG was used to first select a bin for which to search using a weighted choice, and then again to select a pseudorandom address in the bin to probe. This process was described in more detail in Chapter 3

A pilot study was performed for the `MonteCarlo-2` experiment prior to commencing data collection. The pilot study for the `MonteCarlo-2` experiment tested a sample of 10,000 pseudorandomly chosen probes across the target address space. Figure 4-26 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the Monte Carlo search algorithm probed target addresses during the `MonteCarlo-2` sub-experiments. Like the `MonteCarlo-1` pilot study, it is evident from Figure 4-26 that there are patterns in the distribution of probes across the address space. Again, for the purposes of this experiment these traits have little impact since the aim of this experiment was to achieve a near-uniform distribution of probes across the address



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-24: Results of the Random Monte Carlo search against the randomised dataset (sub-experiment `MonteCarlo-1a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-25: Results of the Random Monte Carlo search against the surveyed dataset (sub-experiment `MonteCarlo-1b`) highlighting the number of valid probes delivered and process times per simulation.

space for which the pilot study demonstrates.

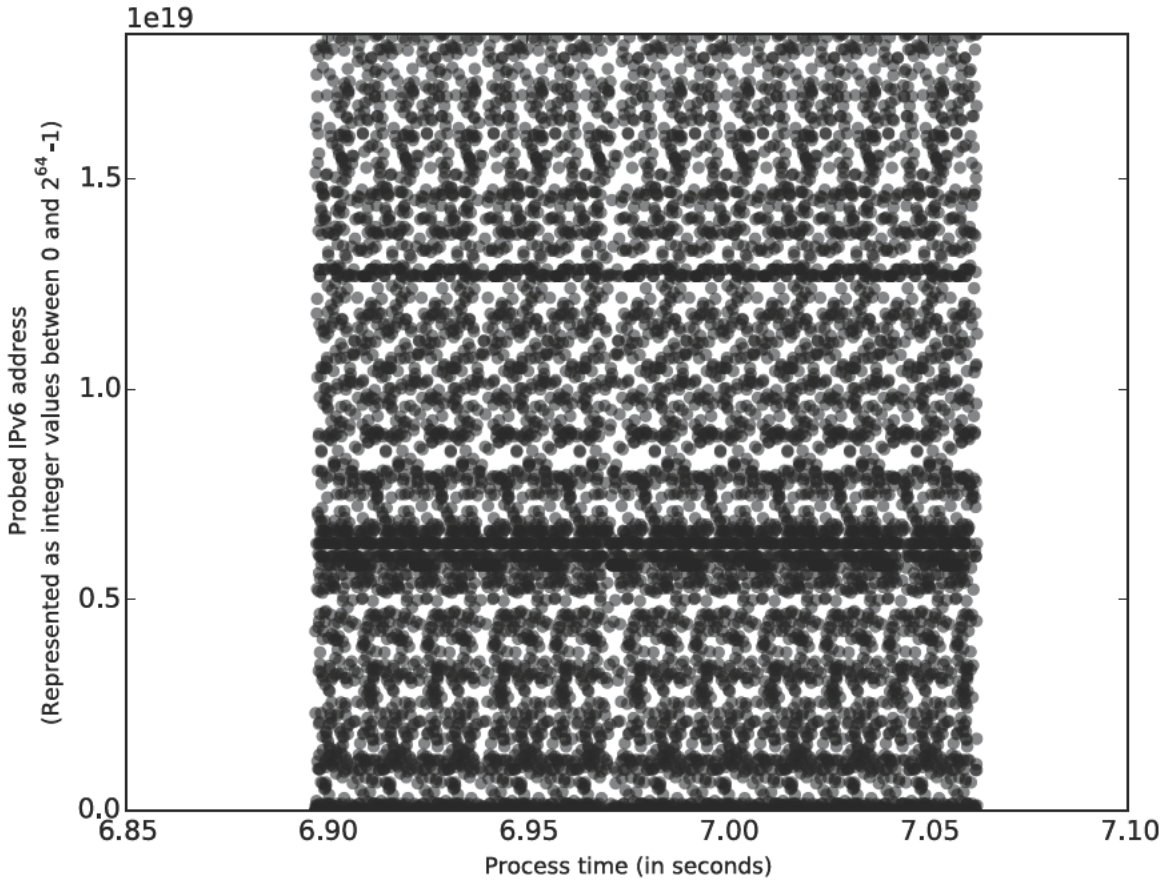


Figure 4-26: Probe distribution for the `MonteCarlo-2` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

Each simulation probed 2^{32} pseudorandomly selected addresses from a starting point selected through the means described above. The results for the weighted `MonteCarlo-2a` and `MonteCarlo-2b` sub-experiments are included in Table 4.12.

Unlike `MonteCarlo-1`, the `MonteCarlo-2` experiment recorded successful hits in both sub-experiments. A mean number of successful hits of 1.2 was recorded for the `MonteCarlo-2a` experiment (see Figure 4-27(a)). Although the minimum recorded successful probes during a simulation was observed to be 0, the maximum was 4. Overall the sub-experiment recorded successful probes against 119 unique IPv6 hosts. A histogram of the successfully probed hosts across the most observed buckets of the address space is included in Figure 4-29. It is clear from Figure 4-29 that no more than a single address was probed per bucket.

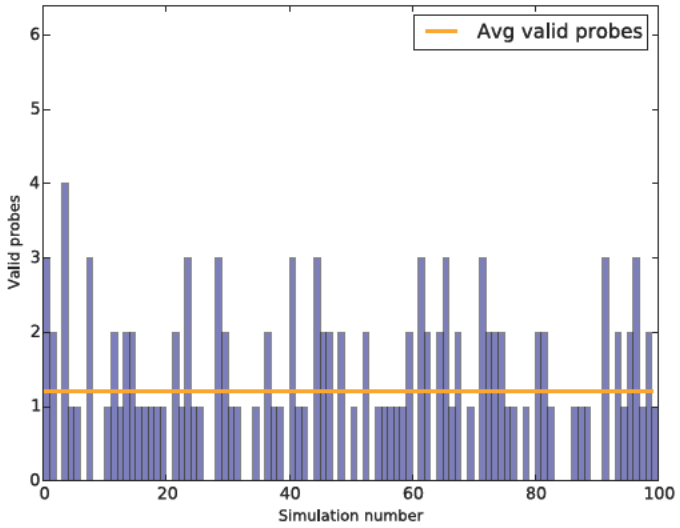
`MonteCarlo-2b` faired better, with an average number of valid probes registered at 30.42 (see Figure 4-28(a)) amongst a maximum of 45, and a minimum of 15. Overall the `MonteCarlo-2b` sub-experiment observed 2,914 unique hosts on the IPv6 network, which is amongst the highest values recorded. The distribution of observed hosts across the bucketed address space is included in Figure 4-30. From Figure 4-30 it is clear that the majority of the uniquely discovered hosts lie in the first bucket of the address space.

To determine whether the results of the two sub-experiments were significantly different, the Wilcoxon Rank-Sum test was applied. The test statistic was $W = 0.0$, and the p-value was $p =< 0.001$ with a target α value of $\alpha = 0.05$. This highlights a significant difference between the ranked means of sub-experiment `MonteCarlo-2a` and sub-experiment `MonteCarlo-2b`.

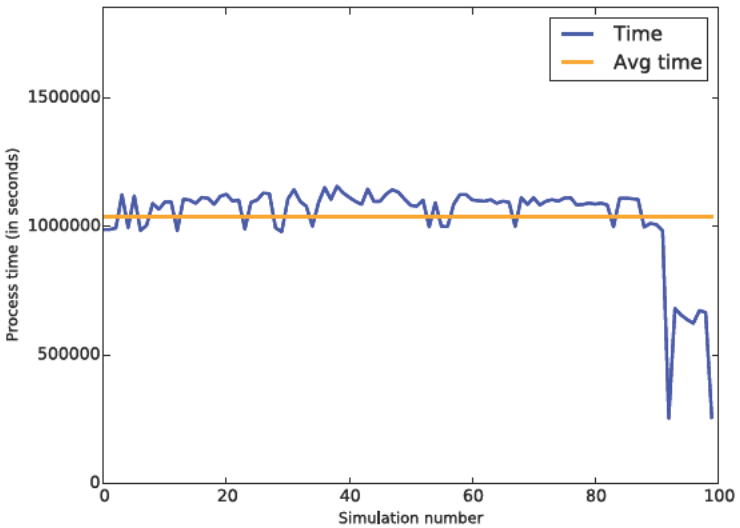
The average process completion times for the `Monte_Carlo-2` experiment were the highest recorded, and were substantially higher than any other experiment conducted. `Monte_Carlo-2a` recorded a mean process completion time of 1,038,336 seconds, or approximately 288.5 hours (see Figure 4-24(a)). The `Monte_Carlo-2b` sub-experiment recorded a similar mean process completion time of 1,318,608 seconds, or approximately 366 hours (see Figure 4-25(a)).

Table 4.12: The descriptive statistics of the results gathered from the Weighted Monte Carlo simulations (experiments `MonteCarlo-2a` and `MonteCarlo-2b`).

Experimental parameters		Experiment number	
		MonteCarlo-2a	MonteCarlo-2b
Valid probes		(/50000)	(/41171)
	Maximum	4 (0.0080%)	45 (0.1093%)
	Mean	1.20 (0.0024%)	30.42 (0.0739%)
	Minimum	0 (0.0000%)	15 (0.0364%)
	Median	1.0	30.0
	Total unique hosts discovered	119	2914
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	1155906.63	1449487.21
	Mean	1038336.41	1318608.34
	Minimum	251762.16	662446.69

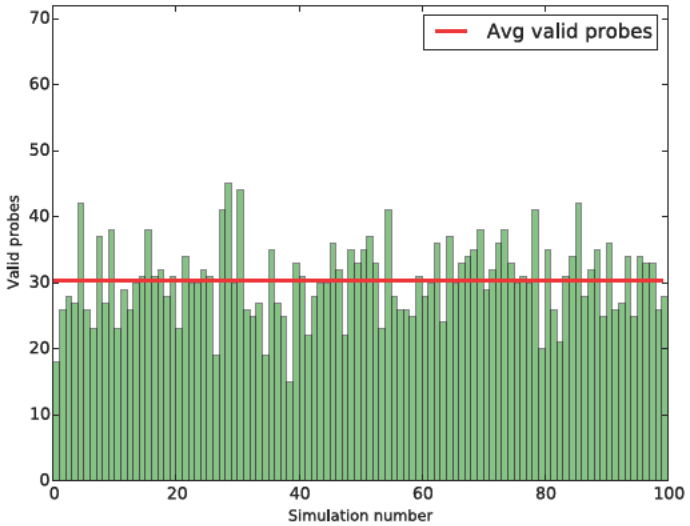


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

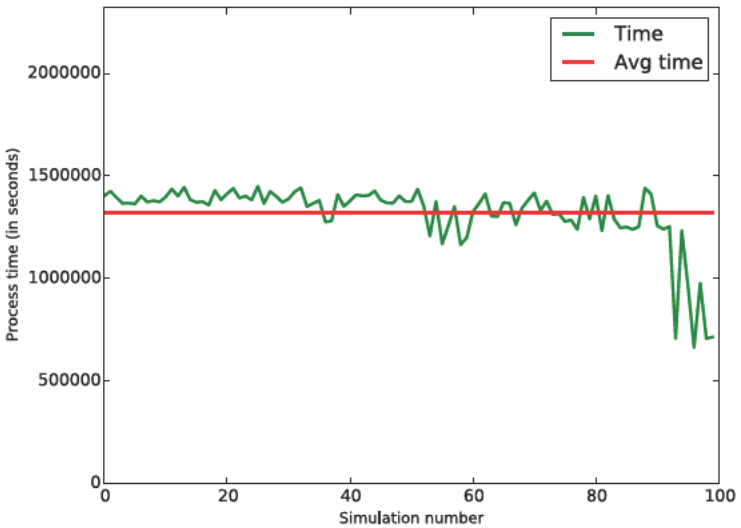


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-27: Results of the Weighted Monte Carlo search against the randomised dataset (sub-experiment `MonteCarlo-2a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-28: Results of the Weighted Monte Carlo search against the surveyed dataset (sub-experiment `MonteCarlo-2b`) highlighting the number of valid probes delivered and process times per simulation.

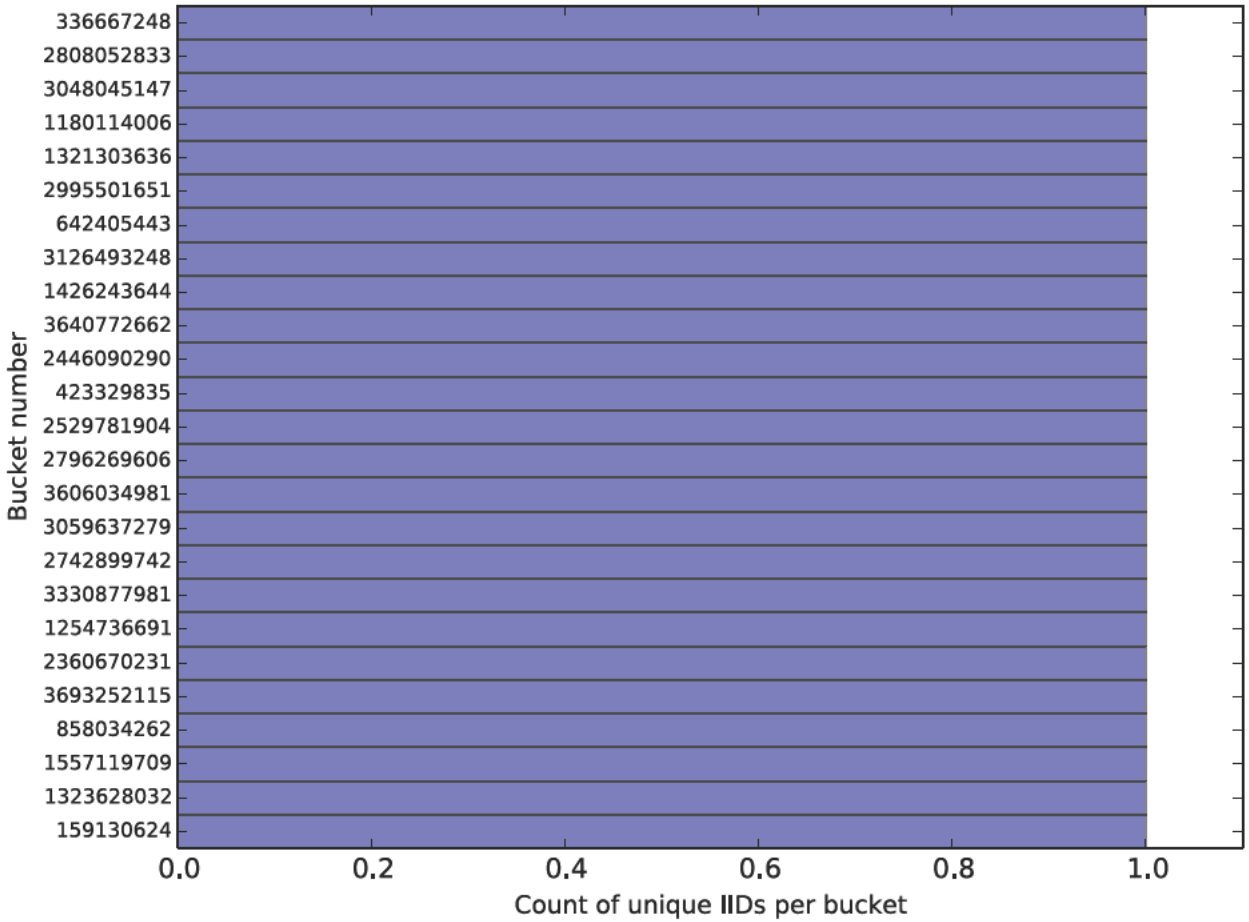


Figure 4-29: The frequency of unique identified nodes across all simulations in the `MonteCarlo-2a` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

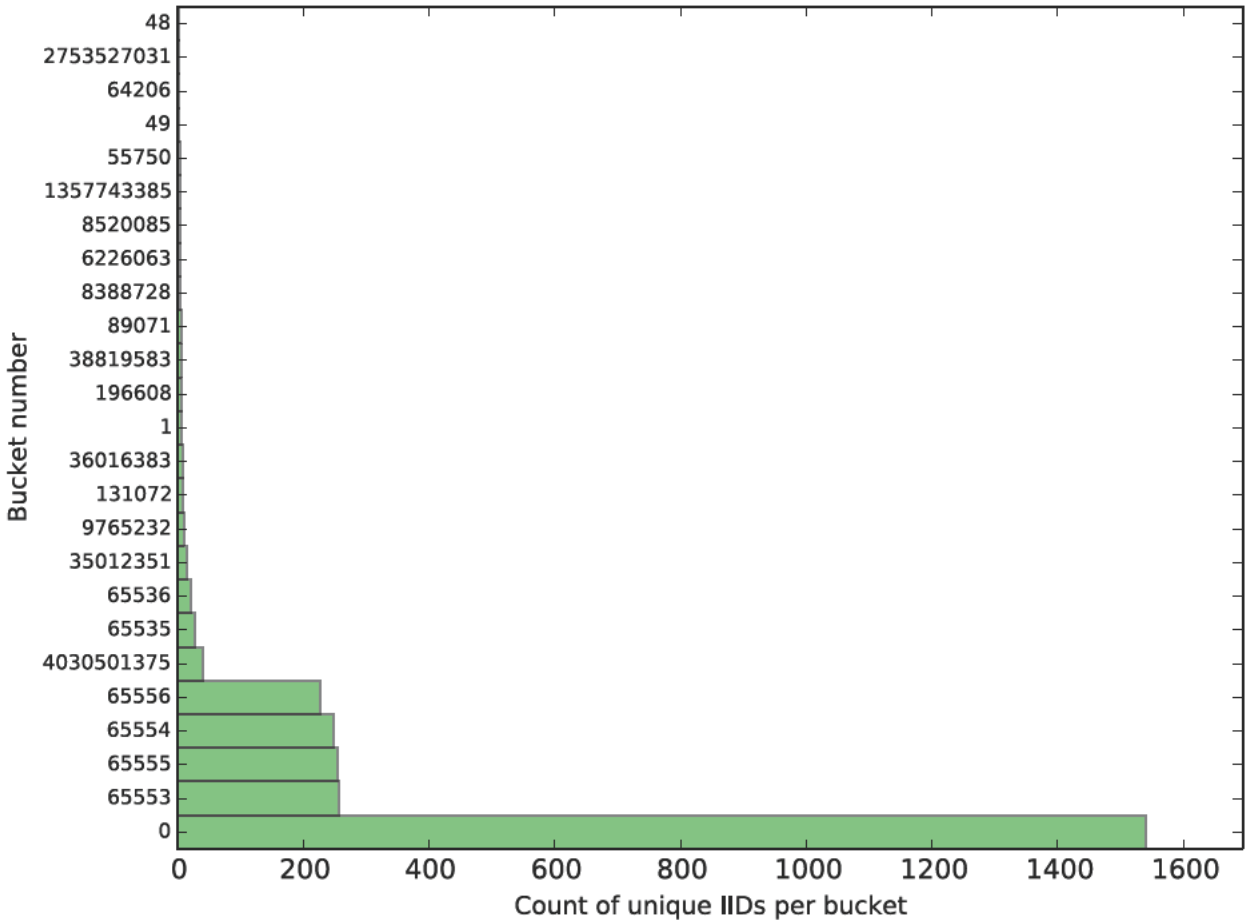


Figure 4-30: The frequency of unique identified nodes across all simulations in the `MonteCarlo-2b` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

4.6 GA search

The GA search algorithm was devised to test whether machine learning methods could be applied to IPv6 host enumeration. Specifically the GA was chosen as a candidate algorithm since it provided a means to combine clustered and randomised address generation strategies. The genetic algorithm was tested during three major experiments; the `GA-1` experiment, which tested the algorithm using an initial population devised of zeroes; the `GA-2` experiment, which tested the algorithm using a pseudorandomly generated starting population as the basis for the genetic process, and the `GA-3` experiment; which used a weighted choice to generate parent pairs for selected bins in the address space as described in Chapter 3.

The GA differed from the other experiments since the experimental process involved the use of programs written in C, rather than Python. This difference is evident in the average processing times of the simulations, which is lower on average than the experiments involving Python programs. The C program `ga` was used to implement the GA search algorithm. A wrapper program written in Python (titled `ga_run.py`) was used to configure the `ga` program to deliver a maximum of 2^{32} probes, per simulation, to IPv6 addresses aligning with the specifications detailed above. `ga_run.py` was also configured to perform 100 simulations for each sub-experiment. The outcomes of these experiments are detailed below.

4.6.1 Experiment GA-1: Zero-origin GA search

The zero-origin `GA-1` experiment tested the GA search algorithm against the randomised and surveyed datasets in sub-experiments `GA-1a` and `GA-1b` respectively. For this experiment the `ga` program was configured to generate an initial population of zero-value organisms (i.e. a population of `::` value IPv6 addresses).

A pilot study was performed prior to commencing data collection for the `GA-1` experiment. The pilot study for the `GA-1` experiment tested a sample of 10,000 generated organisms. Figure 4-31 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the GA search algorithm probed target addresses during the `GA-1`

sub-experiments. It is evident from Figure 4-31 that clusters of addresses throughout the address space were being targeted. This behaviour was anticipated and desirable for the operation of the search algorithm.

Each simulation probed 2^{32} addresses from an initial population generated by the GA search algorithm through the means described above. The results for the zero-origin `GA-1a` and `GA-1b` sub-experiments are included in Table 4.13. The `GA-1a` recorded a mean number of successful hits of 0 (see Figure 4-33(a)). The `GA-1b` was more successful having recorded a mean number of successful hits of 6.95 (see Figure 4-34(a)). From `GA-1b` a maximum of 12 successful hits and a minimum of 3 successful hits were recorded across all simulations. Overall the sub-experiment recorded successful probes against 156 unique IPv6 hosts. A histogram of the successfully probed hosts across the most observed buckets of the address space is included in Figure 4-32. It can be seen in Figure 4-32 that the located hosts were predominantly located in the first bucket of the address space.

The Wilcoxon Rank-Sum test was applied to the results of the `GA-1a` and `GA-1b` sub-experiments to determine whether a there was a significant difference between the populations. This test revealed that a significant difference existed between the groups, with a test statistic of $W = 0.0$, and the p-value was $p =< 0.001$ with a target α value of $\alpha = 0.05$. This highlights that there was a significant difference between the ranked means of sub-experiment `GA-1a` and sub-experiment `GA-1b`.

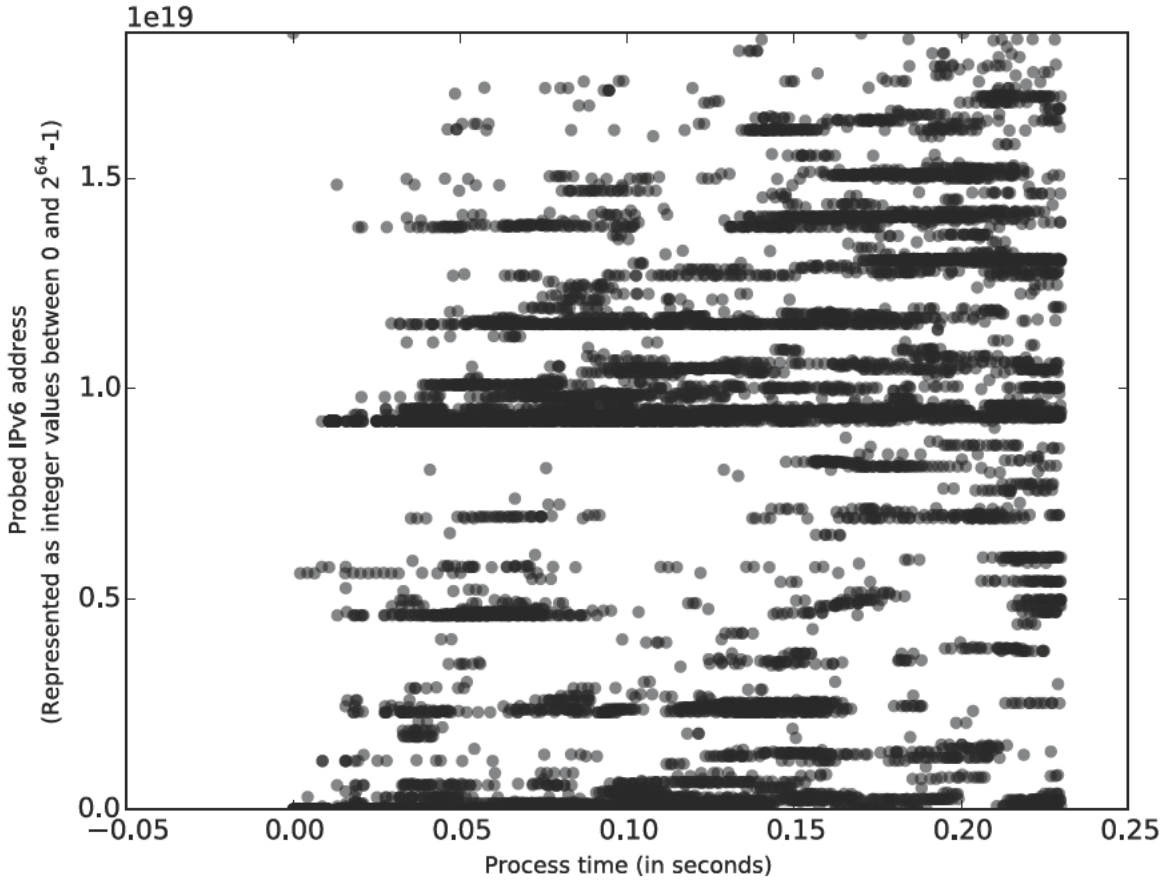


Figure 4-31: Probe distribution for the `GA-1` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

Table 4.13: The descriptive statistics of the results gathered from the Zero Origin GA simulations (experiments `GA-1a` and `GA-1b`).

Experimental parameters		Experiment number	
		GA-1a	GA-1b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	12 (0.0291%)
	Mean	0.00 (0.0000%)	6.95 (0.0169%)
	Minimum	0 (0.0000%)	3 (0.0073%)
	Median	0.0	7.0
	Total unique hosts discovered	0	156
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	14866.01	14784.47
	Mean	13985.10	13910.58
	Minimum	12368.09	12283.55

The average process completion times for the `GA-1` experiment were amongst the lowest recorded. `GA-1a` recorded a mean process completion time of 13985 seconds, or approximately 4 hours (see Figure 4-33(b)). The `GA-1b` sub-experiment recorded a similar mean process completion time of 13,910 seconds, or approximately 4 hours (see Figure 4-34(b)).

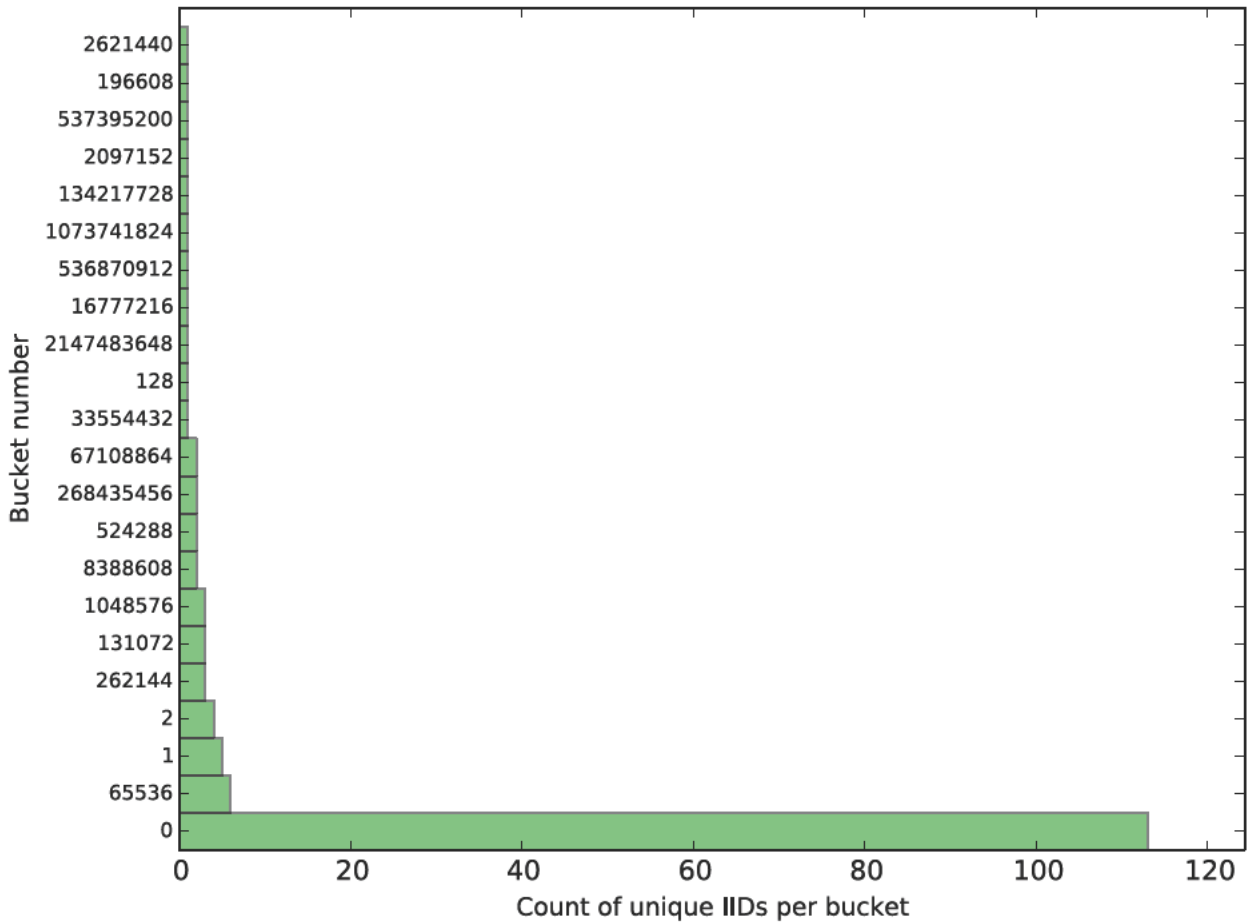
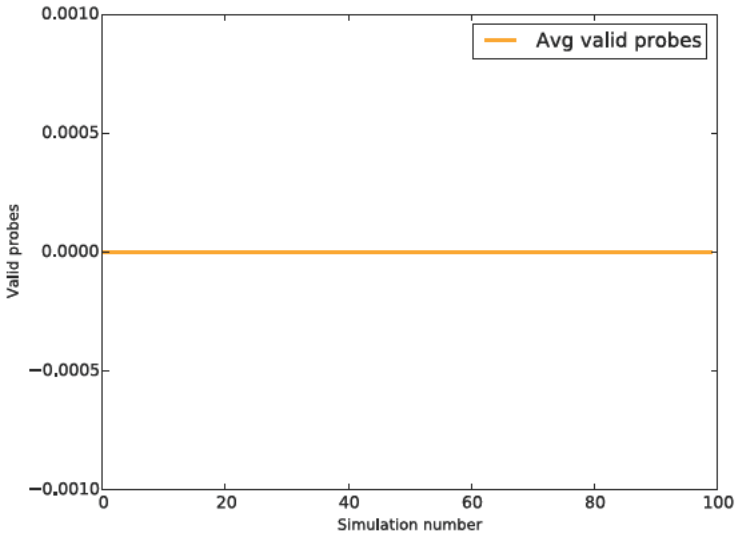


Figure 4-32: The frequency of unique identified nodes across all simulations in the `GA-1b` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

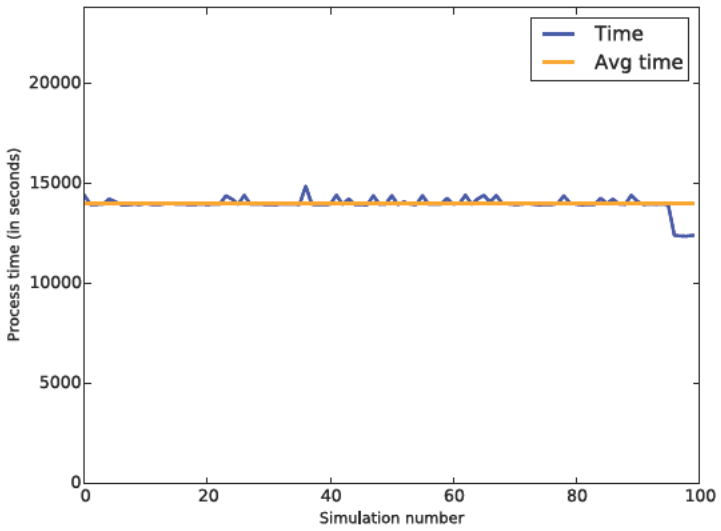
4.6.2 Experiment GA-2: Stochastic GA search

The stochastic `GA-2` experiment tested the GA search algorithm against the randomised and surveyed datasets in sub-experiments `GA-2a` and `GA-2b` respectively. For this experiment the `ga` program was configured to generate an initial population of pseudorandomly generated organisms (i.e. a population of IPv6 addresses between 0 and $2^{64} - 1$).

A pilot study was performed prior to commencing data collection for the `GA-2` experiment. The pilot study for the `GA-2` experiment tested a sample of 10,000 generated organisms. Figure 4-35 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the GA search algorithm probed target addresses during the `GA-2` sub-experiments. Like `GA-1`, it is evident from Figure 4-35 that clusters of addresses throughout the address space were being targeted. This behaviour was, again, anticipated and desirable for the operation of the search algorithm.

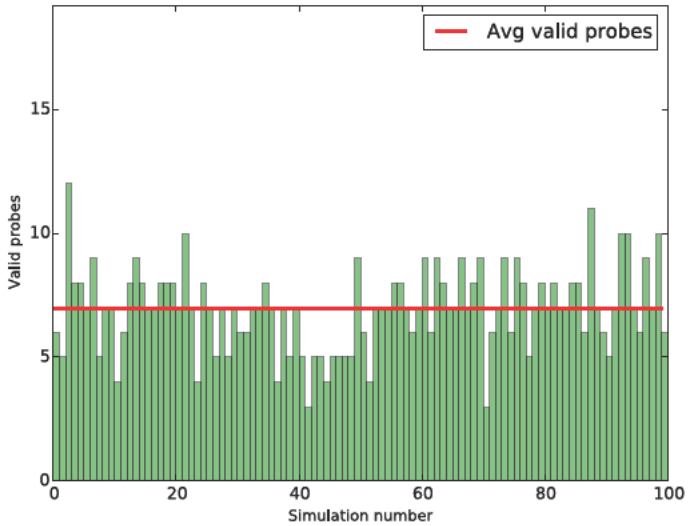


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

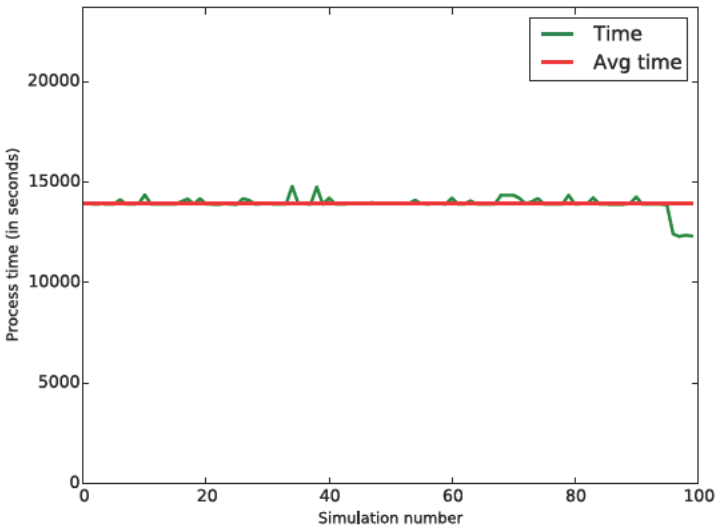


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-33: Results of the ZeroOrigin GA search against the randomised dataset (sub-experiment `GA-1a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-34: Results of the ZeroOrigin GA search against the surveyed dataset (sub-experiment `GA-1b`) highlighting the number of valid probes delivered and process times per simulation.

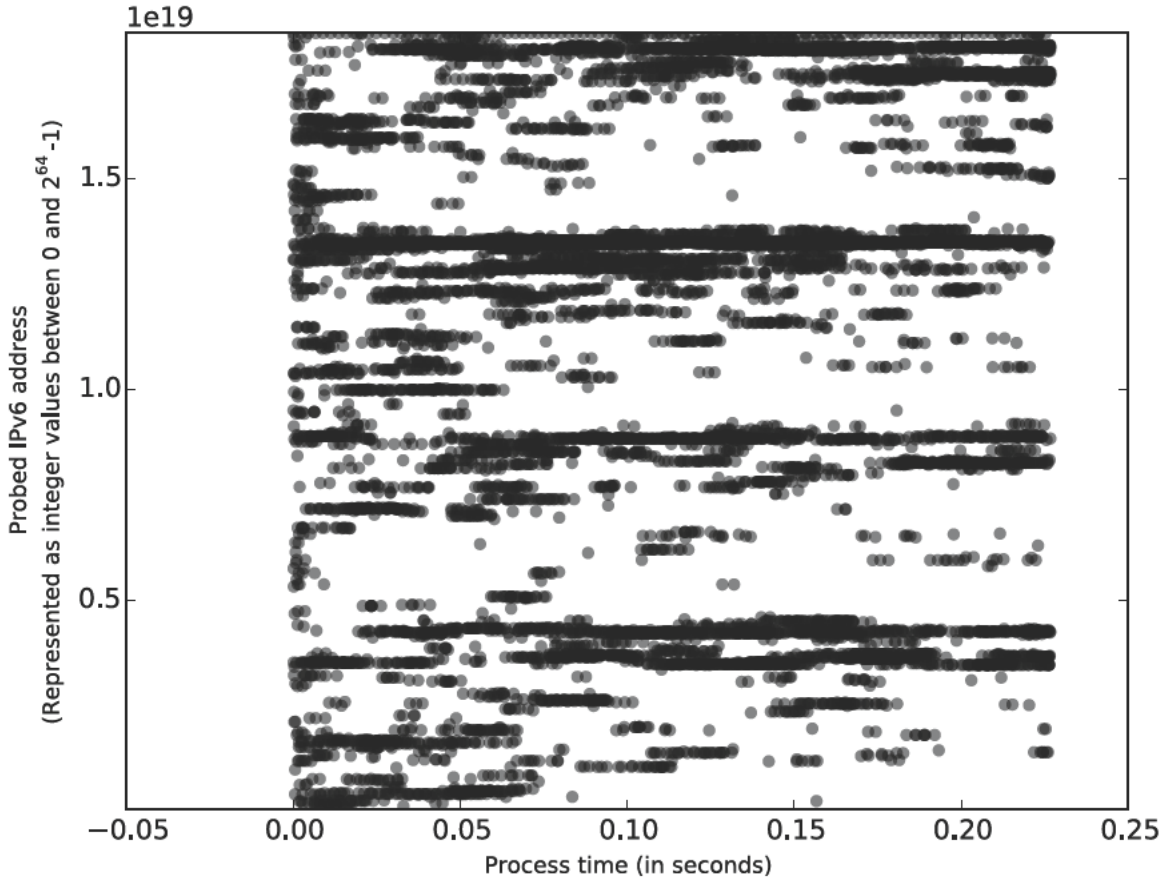


Figure 4-35: Probe distribution for the `GA-2` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

Each simulation probed 2^{32} addresses from an initial population of organisms that were pseudorandomly generated by a PRNG. The results for the stochastic `GA-2a` and `GA-2b` sub-experiments are included in Table 4.14. Unfortunately both sub-experiments `GA-2a` and `GA-2b` recorded a mean number of successful hits of 0. Since both of the sub-experiments revealed counts of zero, the samples of the populations were identical, and therefore no significant difference existed between the sets (i.e. $\mu = \mu_0$).

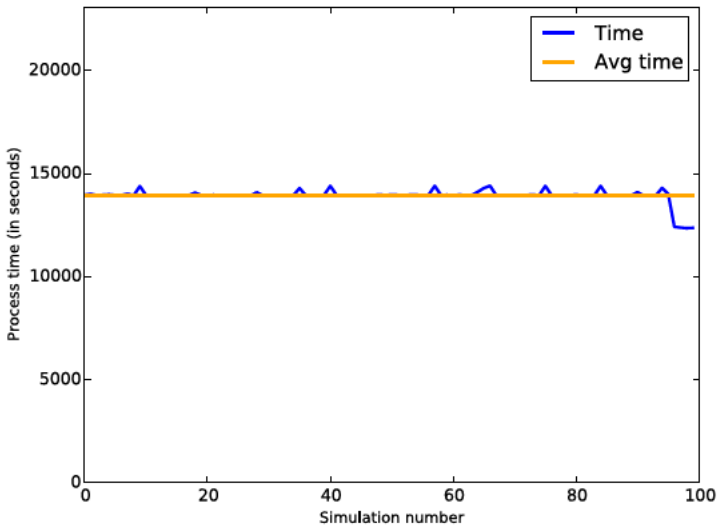
Table 4.14: The descriptive statistics of the results gathered from Stochastic GA simulations (experiments `GA-2a` and `GA-2b`).

Experimental parameters		Experiment number	
		GA-2a	GA-2b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	0 (0.0000%)
	Mean	0.00 (0.0000%)	0.00 (0.0000%)
	Minimum	0 (0.0000%)	0 (0.0000%)
	Median	0.0	0.0
	Total unique hosts discovered	0	0
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	14412.07	14537.89
	Mean	13938.12	13924.74
	Minimum	12353.20	12288.42

The average process completion times for the `GA-2` experiment were very similar to that of `GA-1`, and were again amongst the lowest recorded. `GA-2a` recorded a mean process completion time of 13,938 seconds, or approximately 4 hours (see Figure 4-36(a)). The `GA-2b` sub-experiment recorded a similar mean process completion time of 13,924 seconds, or approximately 4 hours (see Figure 4-37(a)).

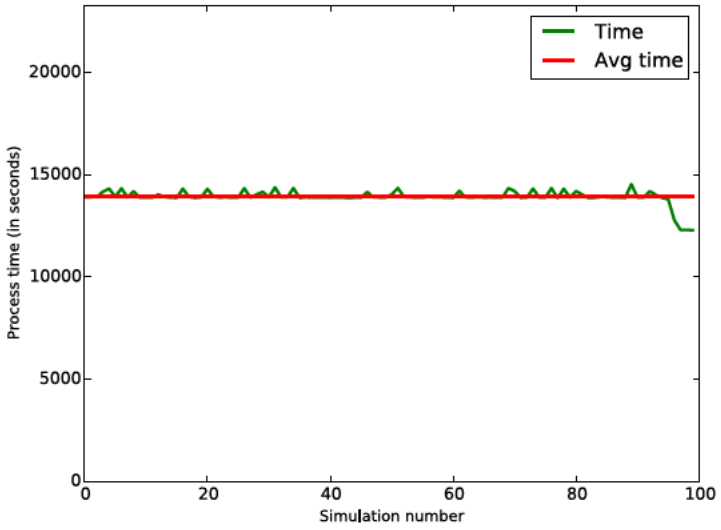
4.6.3 Experiment GA-3: Weighted GA search

The weighted `GA-3` experiment tested the GA search algorithm against the randomised and surveyed datasets in sub-experiments `GA-3a` and `GA-3b` respectively. For this experiment the `ga` program was configured to generate an initial population of using



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-36: Results of the Random GA search against the randomised dataset (sub-experiment `GA-2a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-37: Results of the Stochastic GA search against the surveyed dataset (sub-experiment `GA-2b`) highlighting the number of valid probes delivered and process times per simulation.

a weighted choice process. The process resulted in a population of "parent-pair" addresses that represented the first and last addresses in selected buckets of the 64 bit address space. This process was detailed in Chapter 3.

A pilot study was performed prior to commencing data collection for the `GA-3` experiment. The pilot study for the `GA-3` experiment tested a sample of 10,000 generated organisms. Figure 4-38 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the GA search algorithm probed target addresses during the `GA-3` sub-experiments. It is again evident from Figure 4-38 that clusters of addresses throughout the address space were being targeted. This behaviour was again, anticipated and desirable for the operation of the search algorithm.

Each simulation probed 2^{32} addresses from an initial population of organisms that were generated. The results for the weighted `GA-3a` and `GA-3b` sub-experiments are included in Table 4.15. Like `GA-2`, both sub-experiments `GA-3a` and `GA-3b` recorded a mean number of successful hits of 0. Since both of the sub-experiments revealed counts of zero, the samples of the populations were identical, and therefore no significant difference existed between the sets (i.e. $\mu = \mu_0$).

Table 4.15: The descriptive statistics of the results gathered from Weighted GA simulations (experiments `GA-3a` and `GA-3b`).

Experimental parameters		Experiment number	
		GA-3a	GA-3b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	0 (0.0000%)
	Mean	0.00 (0.0000%)	0.00 (0.0000%)
	Minimum	0 (0.0000%)	0 (0.0000%)
	Median	0.0	0.0
	Total unique hosts discovered	0	0
Total transmitted probes			
	Maximum	4294967296	4294967296
	Mean	4294967296.00	4294967296.00
	Minimum	4294967296	4294967296
	Median	4294967296.0	4294967296.0
Process time (Seconds)			
	Maximum	14032.65	13755.32
	Mean	13267.52	13295.64
	Minimum	11838.93	11805.93

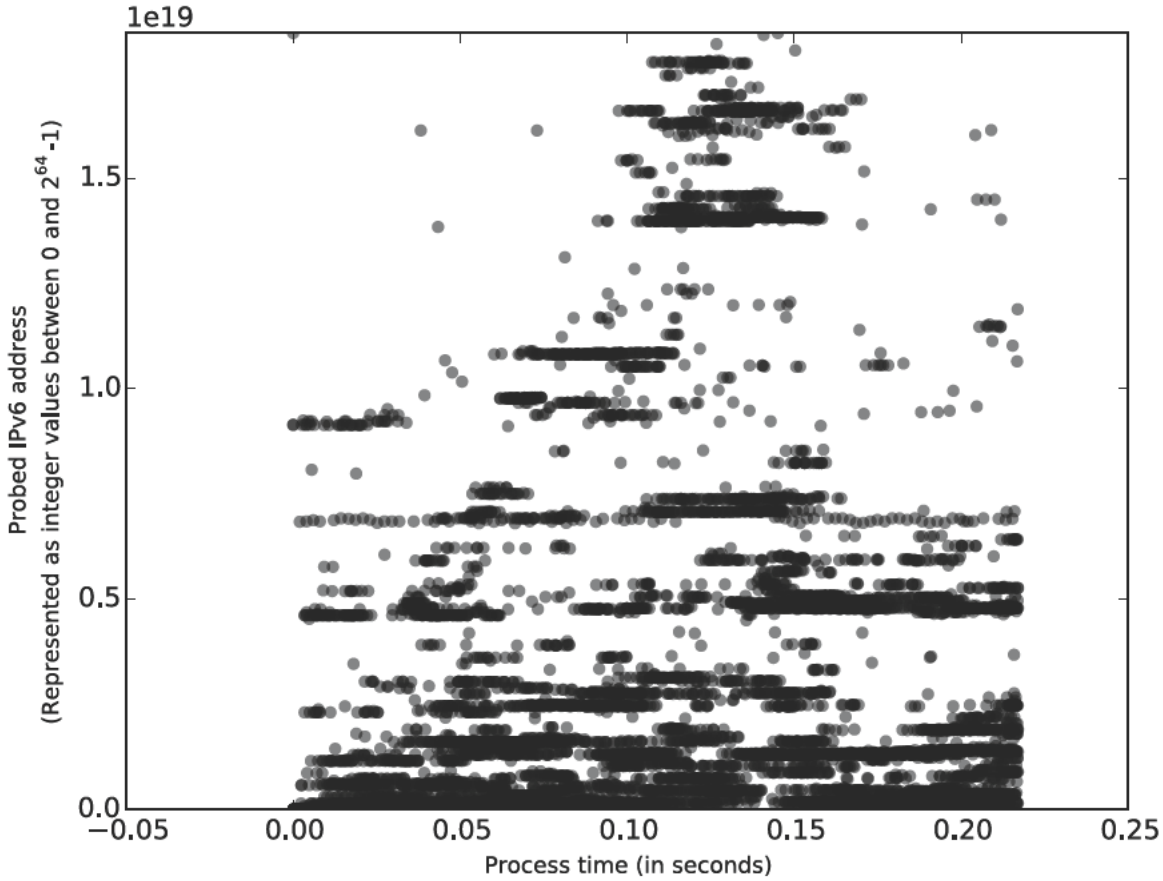
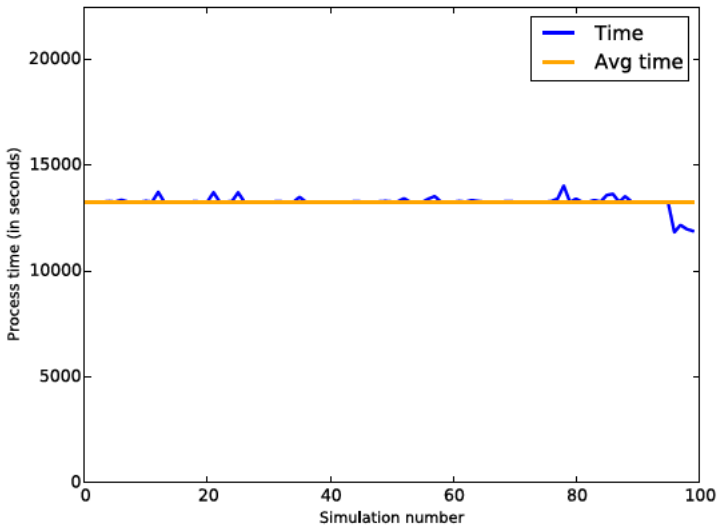


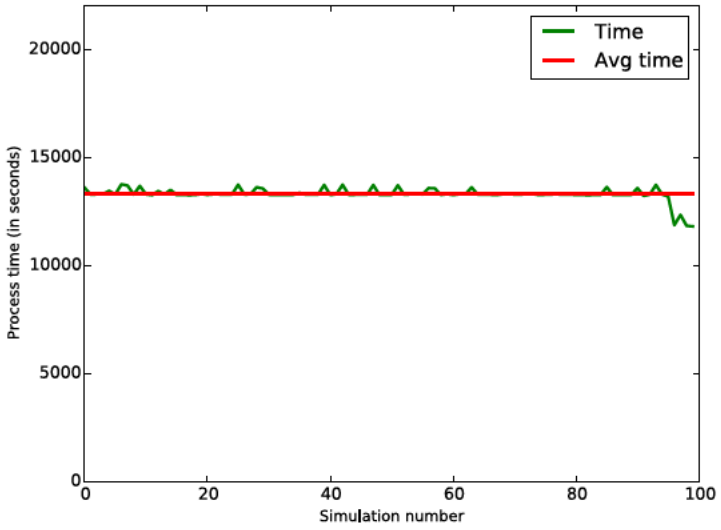
Figure 4-38: Probe distribution for the `GA-3` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

The average process completion times for the `GA-3` experiment were very similar to that of `GA-1` and `GA-2`. `GA-3a` recorded a mean process completion time of 13,267 seconds, or approximately 3.5 hours (see Figure 4-39(a)). The `GA-3b` sub-experiment recorded a similar mean process completion time of 13,295 seconds, or approximately 3.5 hours (see Figure 4-40(a)).



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-39: Results of the Weighted GA search against the randomised dataset (sub-experiment `GA-3a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-40: Results of the Weighted GA search against the surveyed dataset (sub-experiment `GA-3b`) highlighting the number of valid probes delivered and process times per simulation.

4.7 Pattern-based heuristic search

The pattern-based heuristic search algorithm was designed to exploit known patterns in common IPv6 address construction schemes. Similar search techniques have been employed in IPv6 host enumeration tools that are publicly available. This search algorithm tested the pattern-based search algorithm during a single experiment; the `Pattern-1`. This algorithm used a deterministic approach to generating addresses for targeting.

The `pattern.py` program was used to implement the pattern-based heuristic search algorithm. The `pattern.py` program was configured to deliver a maximum of 2^{32} probes, per simulation to IPv6 addresses. `pattern.py` was also configured to perform 100 simulations for each sub-experiment. The outcomes of these experiments are detailed below.

4.7.1 Experiment Pattern-1: Pattern-based heuristic search

The `Pattern-1` experiment tested the pattern-based heuristic search algorithm against the randomised and surveyed datasets in sub-experiments `Pattern-1a` and `Pattern-1b` respectively. For this experiment the `pattern.py` program was configured to generate a sequence of addresses that abused the patterns that exist in IID creation, which are detailed in Carpene and Woodward (2012) and Carpene and Woodward (2014). This process is detailed in Chapter 3.

A pilot study was performed prior to commencing data collection for the `Pattern-1` experiment. The pilot study for the `Pattern-1` experiment tested a sample of 10,000 addresses. Figure 4-41 demonstrates the distribution of probes across the address space over the duration of the pilot study. This graph provides an approximation for how each simulation for the pattern-based heuristic search algorithm probed target addresses during the `Pattern-1` sub-experiments. Figure 4-41 displays a visually sequential search. The first phase of the pattern-based search involves an effectively sequential search, which align with the distribution presented in Figure 4-41.

Each simulation probed 4,284,760,840 addresses. The results for the `Pattern-1a` and `Pattern-1b` sub-experiments are included in Table 4.16. Although `Pattern-1a`

Table 4.16: The descriptive statistics of the results gathered from Pattern simulations (experiments `Pattern-1a` and `Pattern-1b`).

Experimental parameters		Experiment number	
		Pattern-1a	Pattern-1b
Valid probes		(/50000)	(/41171)
	Maximum	0 (0.0000%)	2543 (6.1767%)
	Mean	0.00 (0.0000%)	2543.00 (6.1767%)
	Minimum	0 (0.0000%)	2543 (6.1767%)
	Median	0.0	2543.0
	Total unique hosts discovered	0	2543
Total transmitted probes			
	Maximum	4284760840	4284760840
	Mean	4284760840.00	4284760840.00
	Minimum	4284760840	4284760840
	Median	4284760840.0	4284760840.0
Process time (Seconds)			
	Maximum	261297.31	269453.58
	Mean	247140.80	250607.60
	Minimum	129301.73	133509.73

performed poorly; having recorded 0 successful hits across all simulations (see Figure 4-43(a)), `Pattern-1b` recorded a mean number of successful hits of 2,543 successful hits (see Figure 4-43(a)). The maximum and minimum number of successful hits for `Pattern-1b` were also 2,543. The sub-experiment recorded successful probes against 2,543 unique IPv6 hosts. A histogram of the successfully probed hosts across the most observed buckets of the address space is included in Figure 4-42. It can be seen in Figure 4-42 that almost half of the discovered hosts were located in the first bucket of the address space.

The results of the `Pattern-1a` and `Pattern-1b` sub-experiments were tested for significance using a Wilcoxon Rank-Sum test. This test concluded with a result of $W = 0.0$, and a p-value of $p =< 0.001$ with a target α value of $\alpha = 0.05$. This highlights a significant difference between the ranked means of the two sub-experiments at the 95% confidence interval. Since `Pattern-1b` had a greater count of hosts discovered, it can be concluded that `Pattern-1b`'s results were significantly higher than `Pattern-1a`'s.

The average process completion times for the `Pattern-1` were amongst the highest recorded. `Pattern-1a` recorded a mean process completion time of 247,141 seconds,

or approximately 68.5 hours (see Figure 4-43(b)). The `Pattern-1b` sub-experiment recorded a similar mean process completion time of 250,608 seconds, or approximately 69.5 hours (see Figure 4-44(b)).

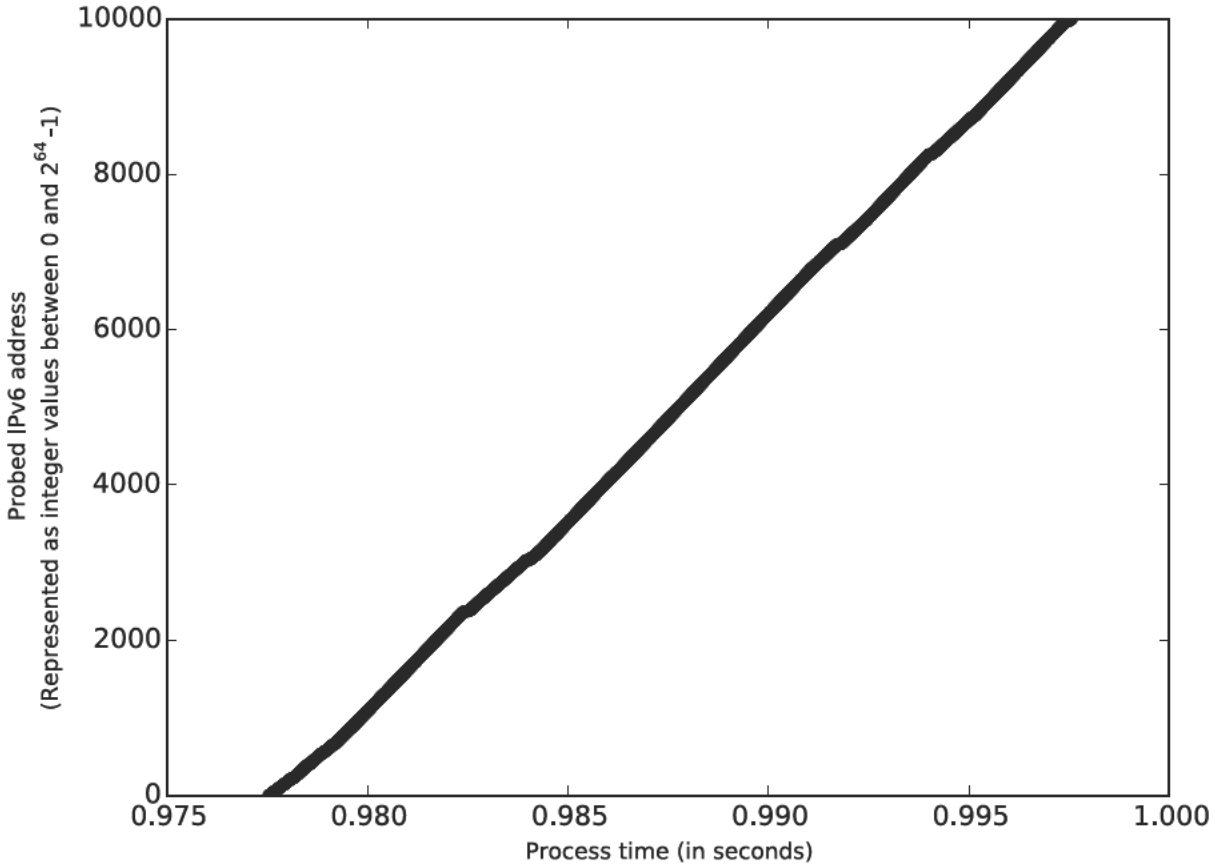


Figure 4-41: Probe distribution for the `Pattern-1` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

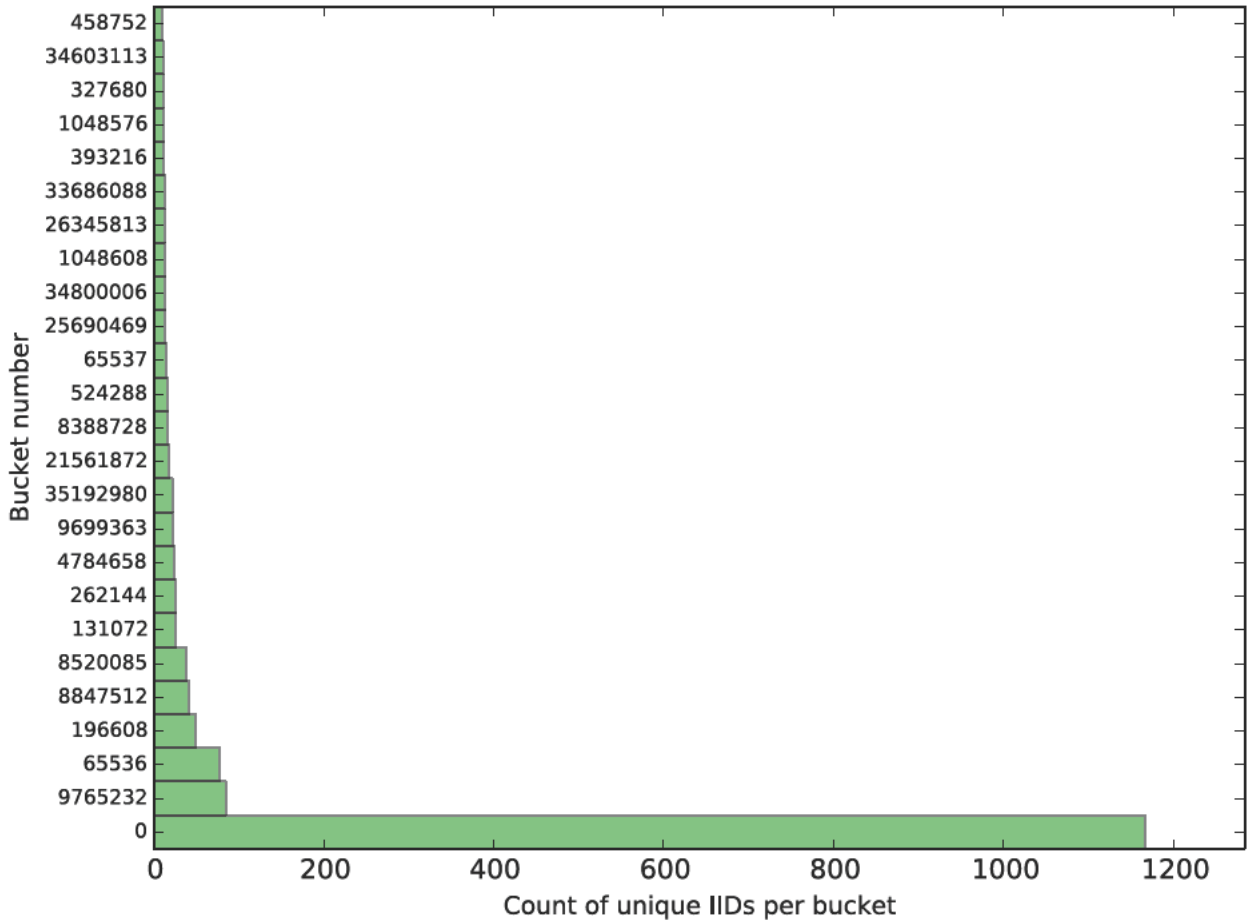
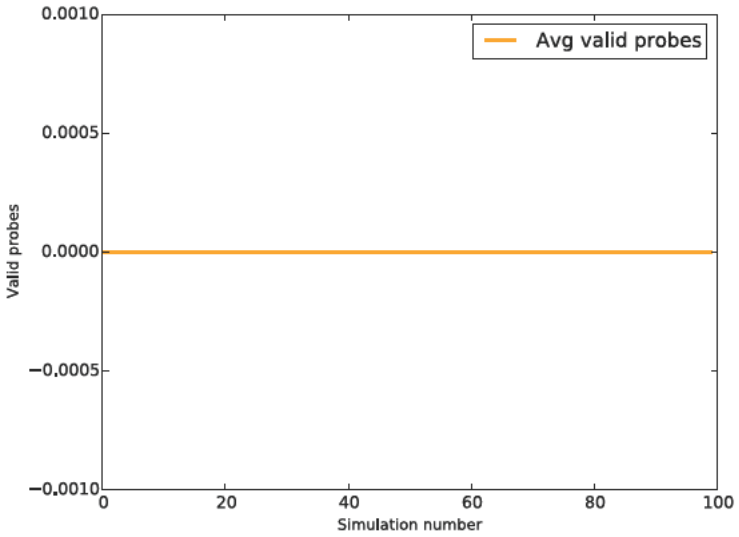
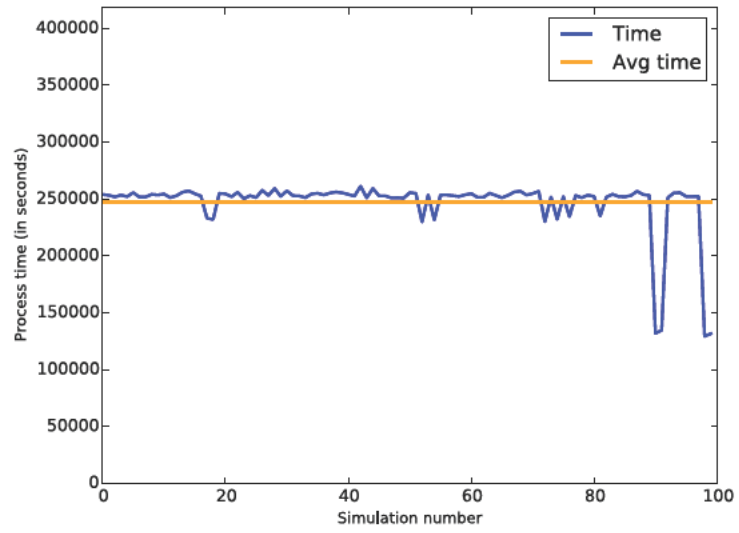


Figure 4-42: The frequency of unique identified nodes across all simulations in the `Pattern-1b` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

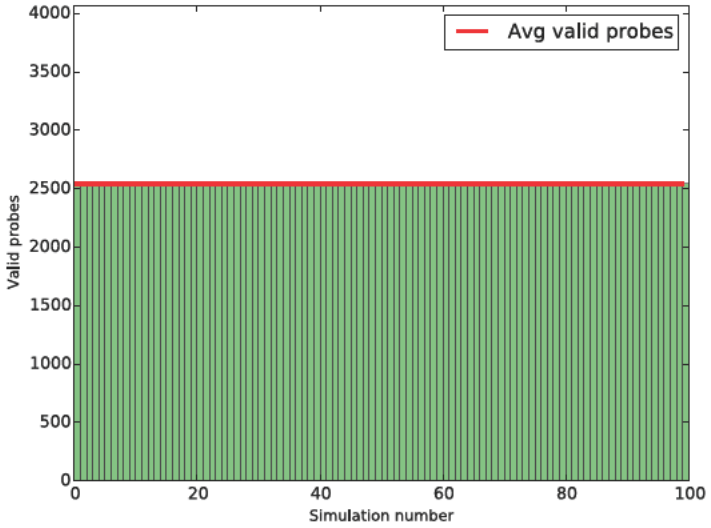


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

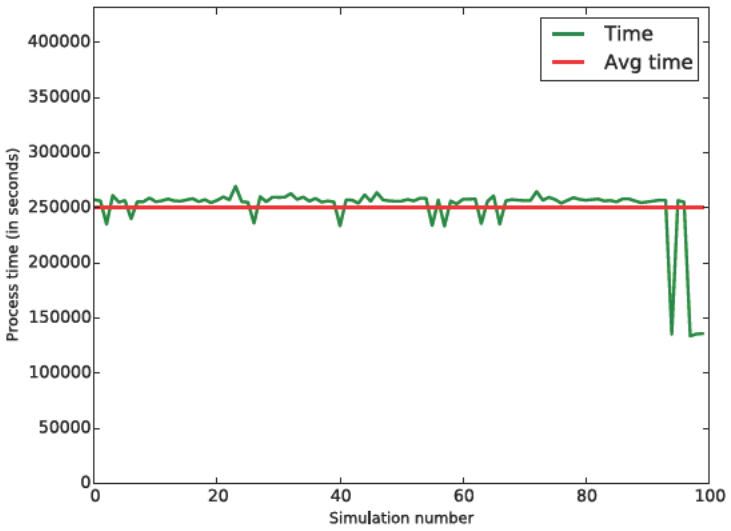


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-43: Results of the Pattern-based search against the randomised dataset (sub-experiment `Pattern-1a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-44: Results of the Pattern-based search against the surveyed dataset (sub-experiment `Pattern-1b`) highlighting the number of valid probes delivered and process times per simulation.

4.8 Adaptive heuristic search

The adaptive heuristic search algorithm was designed explore how machine learning can be applied to IPv6 host enumeration. This algorithm took a multi-faceted approach to the problem, by assessing how discovered addresses have been constructed and then targeting those address creation methods in context. This search strategy has never been applied to the problem thus far. The adaptive search algorithm was tested during a single experiment; the `Adaptive-1`.

The `adaptive.py` program was used to implement the adaptive heuristic search algorithm. The `adaptive.py` program was configured to deliver a maximum of 2^{32} probes, per simulation to IPv6 addresses. `adaptive.py` was also configured to perform 100 simulations for each sub-experiment. The outcomes of these experiments are detailed below.

4.8.1 Experiment Adaptive-1: Adaptive heuristic search

The `Adaptive-1` experiment tested the adaptive heuristic search algorithm against the randomised and surveyed datasets in sub-experiments `Adaptive-1a` and `Adaptive-1b` respectively. For this experiment the `adaptive.py` program was configured to generate a sequence of addresses and probe them. This process is detailed in Chapter 3.

A pilot study was performed prior to commencing data collection for the `Adaptive-1` experiment. The pilot study for the `Adaptive-1` experiment tested a sample of 10,000 addresses. Figure 4-45 demonstrates the distribution of probes across the address space over the duration of the pilot study. Figure 4-45 provides an approximation for how each simulation for the pattern-based heuristic search algorithm probed target addresses during the `Adaptive-1` sub-experiments. It can be observed from Figure 4-45 that address probing appeared to occur in batches. This was a result of the post-hit processing operations of the adaptive search algorithm.

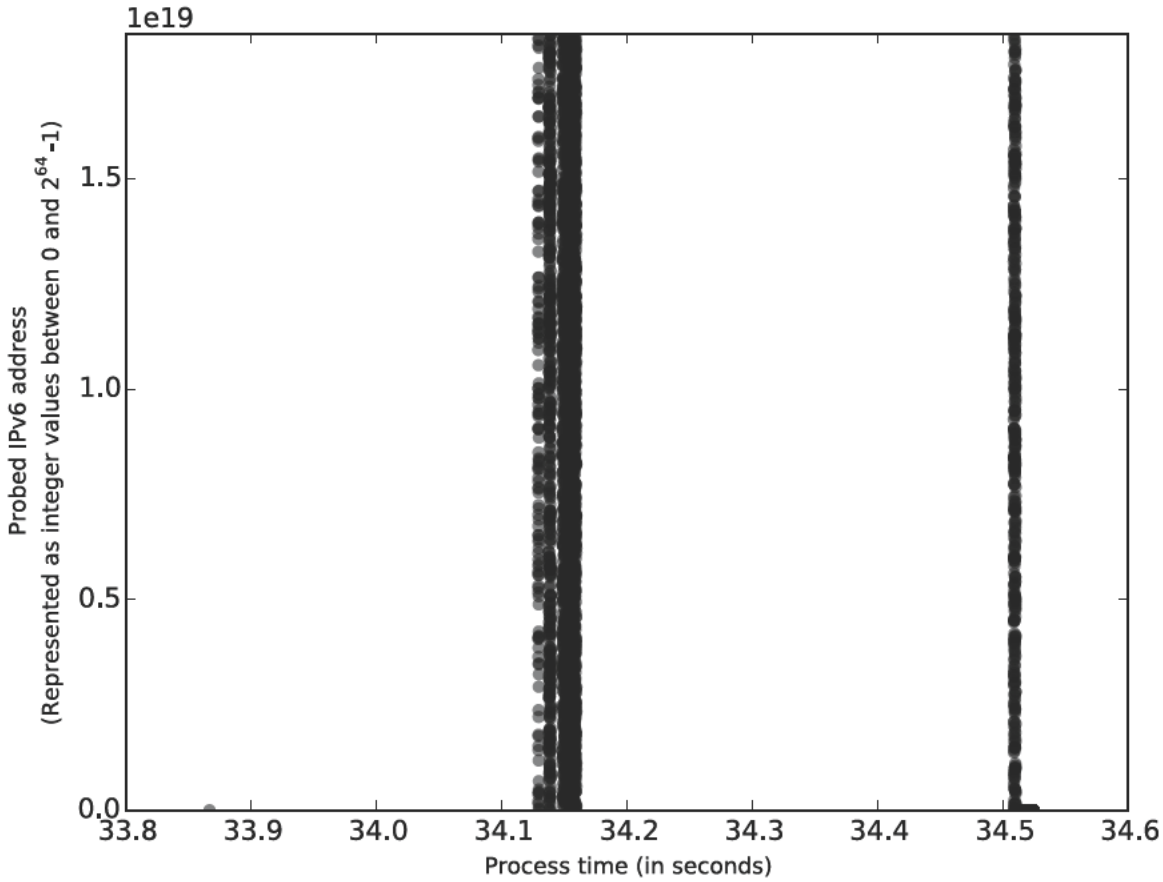


Figure 4-45: Probe distribution for the `Adaptive-1` experiment pilot study. The algorithm's delivery of probes to potential IPv6 addresses is represented over time. The y axis represents the IPv6 addresses that were probed throughout the pilot study as integer values between 0 and $(2^{64} - 1)$. The x axis represents the process time that each probe was delivered.

The results for the `Adaptive-1a` and `Adaptive-1b` sub-experiments are included in Table 4.17. The `Adaptive-1a` experiment probed an average of 6,553,701.6 addresses. This was the lowest recorded average number of probes delivered across all experiments. Of those transmitted probes, an average of 100 successful hits was observed. 100 unique hosts were discovered during `Adaptive-1a`. A histogram of the successfully probed hosts across the most observed buckets of the address space is included in Figure 4-46. It can be seen in Figure 4-46 that almost half of the discovered hosts were located in the first bucket of the address space. Figure 4-46 highlights that `Adaptive-1a` did not see more than a single unique host per bucket discovered.

The `Adaptive-1b` experiment probed a higher average number of addresses than `Adaptive-1a`, with 603,220,298 addresses being probed during this sub-experiment. This was the second lowest recorded average number of probes delivered across all experiments. Of those transmitted probes, an average of 10,218 successful hits was observed. This translated to a total of 10,218 unique hosts being discovered during `Adaptive-1b`. A histogram of the successfully probed hosts across the most observed buckets of the address space is included in Figure 4-47. It can be seen in Figure 4-47 that almost half of the discovered hosts were located in the first bucket of the address space. Figure 4-47 highlights that the majority of unique hosts discovered during `Adaptive-1b` were located within the first bucket of the address space.

The Wilcoxon Rank-Sum test was used to compare the results of the `Adaptive-1` for significance at the 95% confidence interval. This test revealed a significant difference between the two populations, with a test statistic of $W = 0.0$, and a p-value of $p =< 0.001$ with a target α value of $\alpha = 0.05$. This suggests that the results of the `Adaptive-1b` sub-experiment are significantly greater than the results of the `Adaptive-1a` sub-experiment .

The average process completion times for the `Adaptive-1` were some of the lowest recorded. `Adaptive-1a` recorded a mean process completion time of 1,678 seconds, or approximately 0.5 hours which was the lowest recorded process time average across all experiments conducted throughout this research (see Figure 4-48(b)). The `Adaptive-1b` sub-experiment recorded a similar mean process completion time of 20,020 seconds, or approximately 5.5 hours (see Figure 4-49(b)).

Table 4.17: The descriptive statistics of the results gathered from Adaptive simulations (experiments `Adaptive-1a` and `Adaptive-1b`).

Experimental parameters		Experiment number	
		Adaptive-1a	Adaptive-1b
Valid probes		(/50000)	(/41171)
	Maximum	100 (0.2000%)	10218 (24.8184%)
	Mean	100.00 (0.2000%)	10218.00 (24.8184%)
	Minimum	100 (0.2000%)	10218 (24.8184%)
	Median	100.0	10218.0
	Total unique hosts discovered	100	10218
Total transmitted probes			
	Maximum	6553710	603268141
	Mean	6553701.60	603220298.01
	Minimum	6553700	603137066
	Median	6553701.0	603202603.0
Process time (Seconds)			
	Maximum	2004.64	37656.08
	Mean	1637.65	20019.73
	Minimum	186.15	2201.07

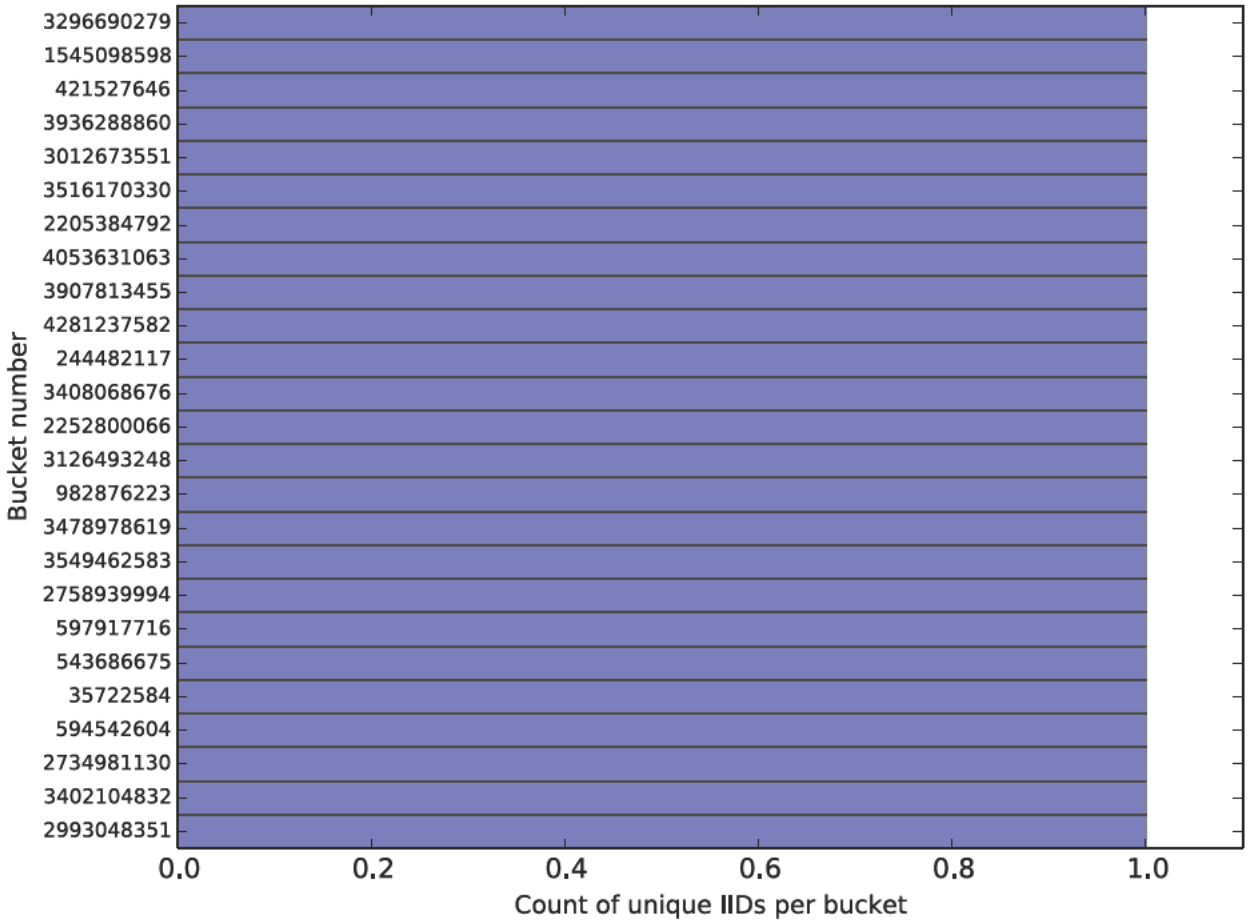


Figure 4-46: The frequency of unique identified nodes across all simulations in the `Adaptive-1a` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

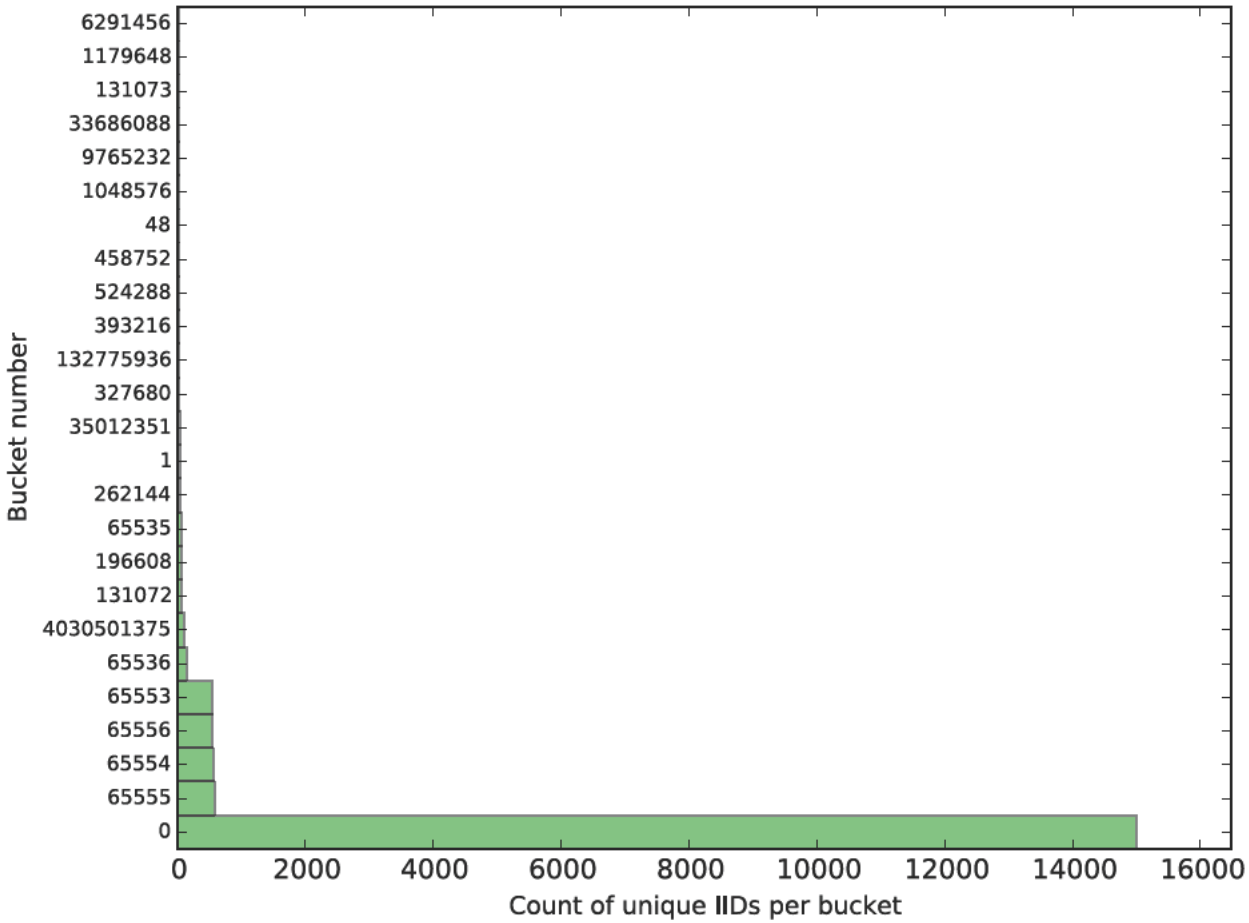
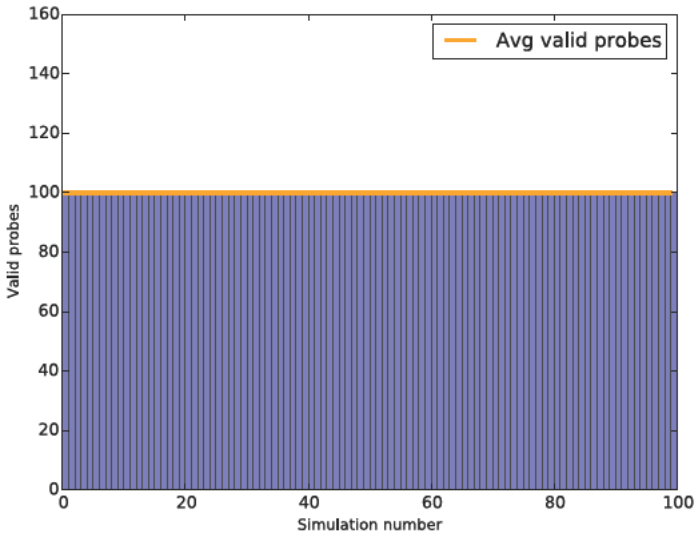
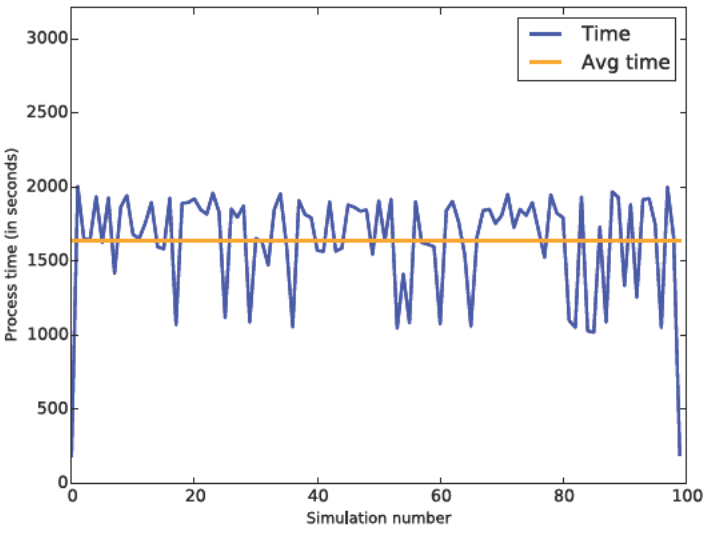


Figure 4-47: The frequency of unique identified nodes across all simulations in the `Adaptive-1b` experiment chunked into 2^{32} buckets with the top 25 observed buckets displayed. The y axis displays the bucket number (between 0 and 2^{32}) whilst the x axis displays the count of unique IIDs observed in the corresponding bucket. The count data was collected across all simulations in the sub-experiment.

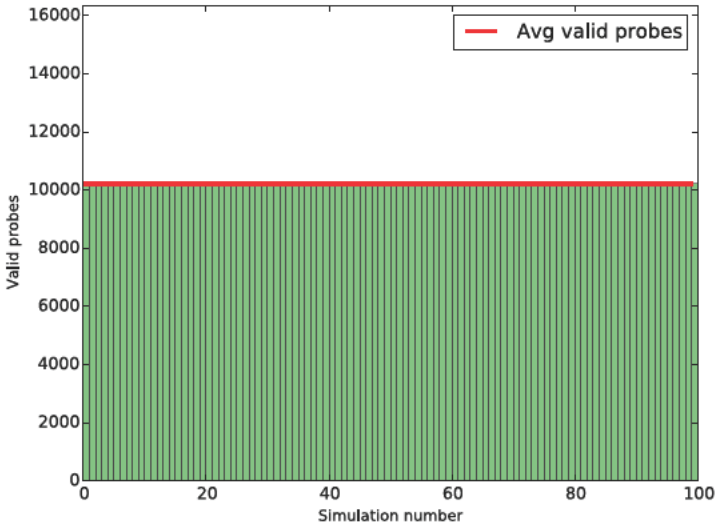


(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot. In this instance both results were identical.

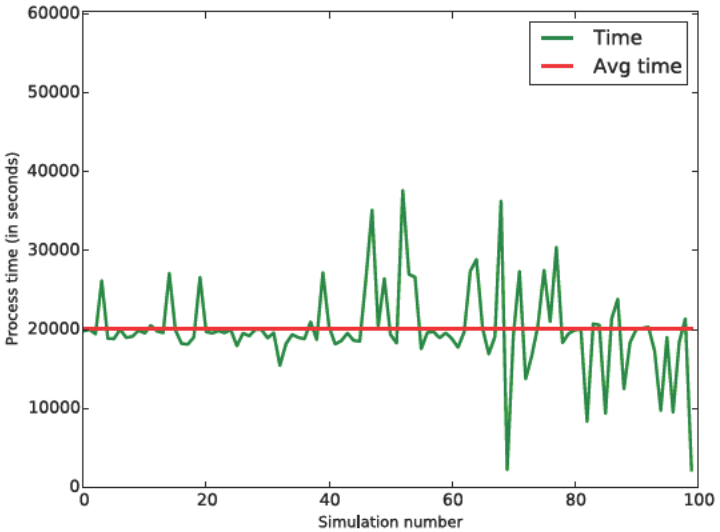


(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-48: Results of the Adaptive search against the randomised dataset (sub-experiment `Adaptive-1a`) highlighting the number of valid probes delivered and process times per simulation.



(a) Number of IPv6 addresses successfully probed per simulation. The average number of successful probes for the sub-experiment is included as a separate plot.



(b) Process time for each simulation performed during the experiment measured in seconds. The average process time for the sub-experiment is included as a separate plot.

Figure 4-49: Results of the Adaptive search against the surveyed dataset (sub-experiment `Adaptive-1b`) highlighting the number of valid probes delivered and process times per simulation.

4.9 Aggregated results

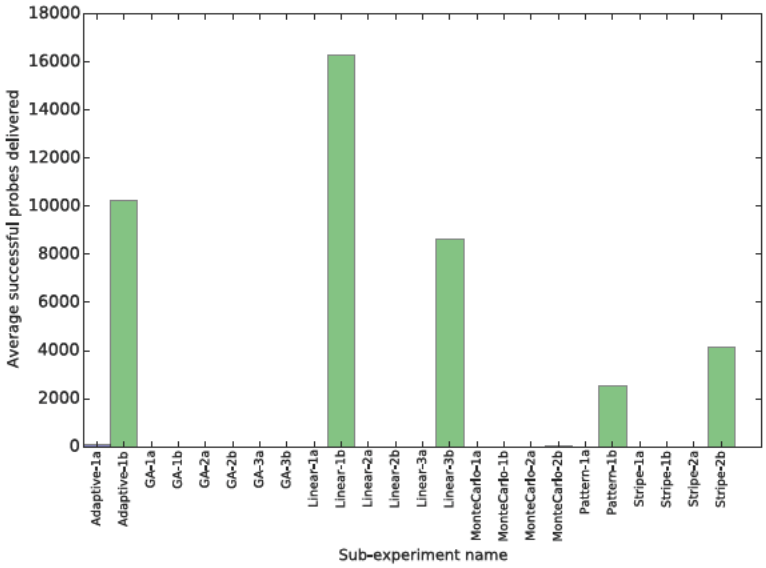
This section provides a summary of the results observed throughout all of the experiments that were conducted. Figure 4-50(a) displays the average successful probes per sub-experiment as a histogram whilst Figure 4-50(b) displays the descriptive statistics of valid hits per experiment as a box and whisker plot. It is evident from the Figure 4-50(a) that the sub-experiment with the greatest number of detected nodes was the `Linear-1b` experiment. Other notable performers include the `Adaptive-1b`, `Linear-3b`, `Stripe-2b` and the `Pattern-1b`. Figure 4-50(b) highlights that the variation between the results recorded per simulation in most of the sub-experiments was limited. The most obvious exception to this is the `Linear-3b` results, which show a stark contrast between the maximum and minimum recorded values.

Figure 4-51(a) displays the average number of delivered probes per sub-experiment as a histogram and Figure 4-51(b) displays the descriptive statistics of the total number of delivered probes per experiment as a box and whisker plot. Figure 4-51(a) highlights that the majority of the simulations that were conducted delivered a similar number of probes. The `Adaptive` sub-experiments were the exception, exhibiting substantially less delivered probes on average per simulation than any of the other experiments. The box and whisker plot for the number of probes delivered (i.e. Figure 4-51(b)) highlights the consistency that was exhibited in delivering probes during each simulation of a sub-experiment.

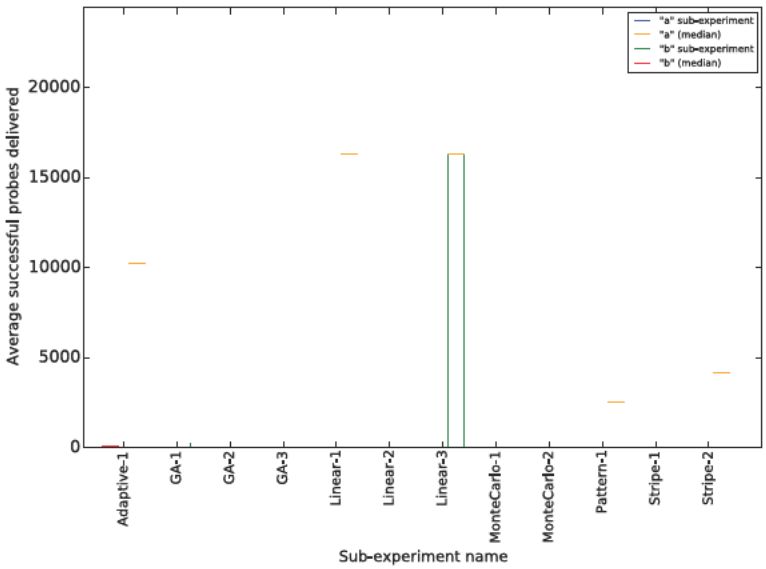
Figure 4-52(a) displays the average process completion time per sub-experiment as a histogram and Figure 4-52(b) displays the descriptive statistics of the process completion times per experiment as a box and whisker plot. Figure 4-52(a) highlights the differences in the process completion times for all of the sub-experiments. It is clear from the histogram that the `MonteCarlo-2` sub-experiments exhibited the highest process times. Contrarily, the `Adaptive` sub-experiments recorded the lowest recorded average process times out of all of the experiments.

To perform the required hypothesis tests for this research, the counts of the valid hosts discovered per simulation were first tested for normality using a Shapiro-Wilks normality test. The results of the Shapiro-Wilks test were $p < 0.000$ with a target

α value of $\alpha = 0.05$, indicating that the null hypothesis for the test (that the data are normally distributed) can be rejected with a high degree of confidence. This implies that the data does not conform to normal distribution. Table 4.18 presents the maximum, minimum and average results for the major dependent variables (number of successful probes, number of transmitted probes, and processing time) for each of the sub-experiments. Table 4.19 provides cumulated totals for the major dependent variables, as well as the total number of simulations conducted, for both of the `a` and `b` sub-experiment types.

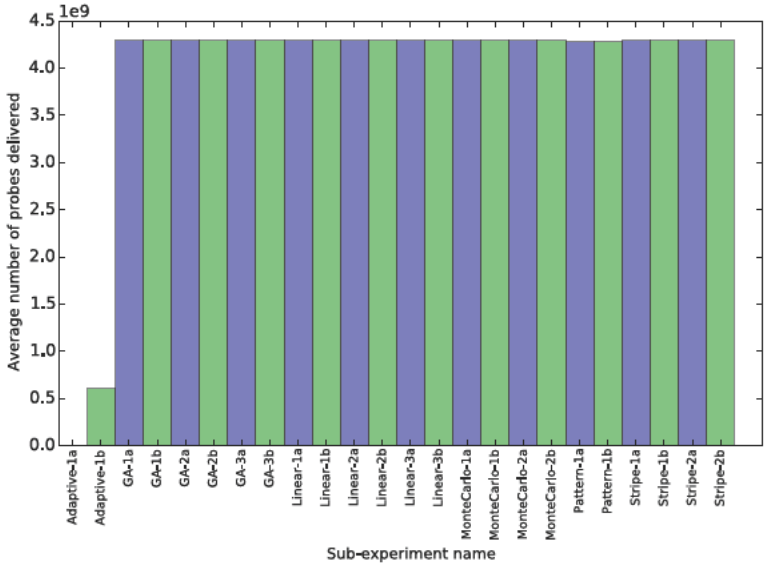


(a) A summary of the mean counts of successful probes for all of the experiments carried out throughout this research. The `a` sub-experiment results are indicated by blue bars, whilst the `b` sub-experiments are denoted by green bars.

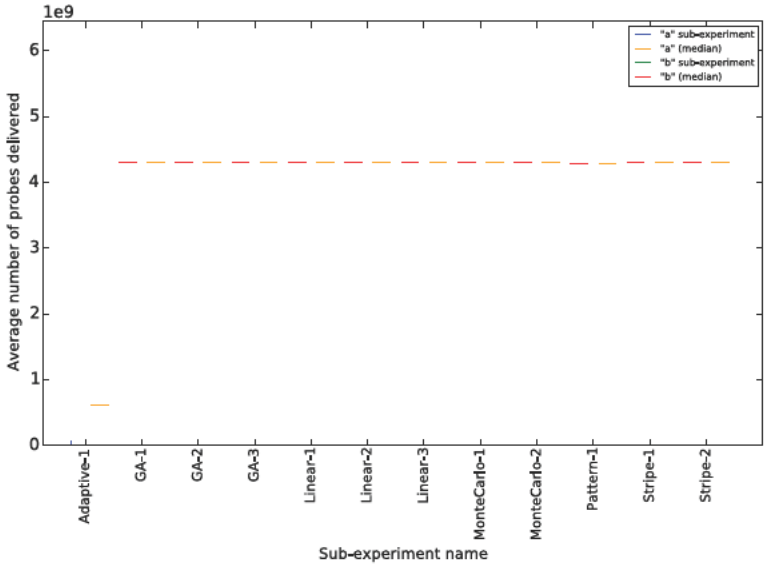


(b) Box and whisker plot depicting the number of successful probes recorded for all experiments. The frequency is shown across the y-axis, whist the x-axis displays the experiments. The `a` sub-experiment results are indicated by blue lines, whilst the `b` sub-experiments are denoted by green lines. Orange and red lines are included to reflect the medians of the `a` and `b` sub-experiments respectively.

Figure 4-50: Summary data of the results of the number of successful probes for the experiments conducted throughout the research.

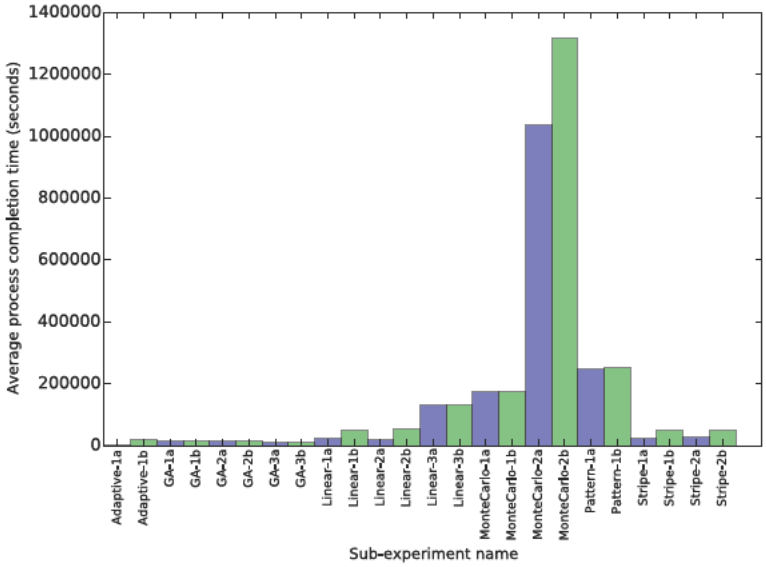


(a) A summary of the mean counts of total delivered probes for all of the experiments carried out throughout this research. The `a` sub-experiment results are indicated by blue bars, whilst the `b` sub-experiments are denoted by green bars.

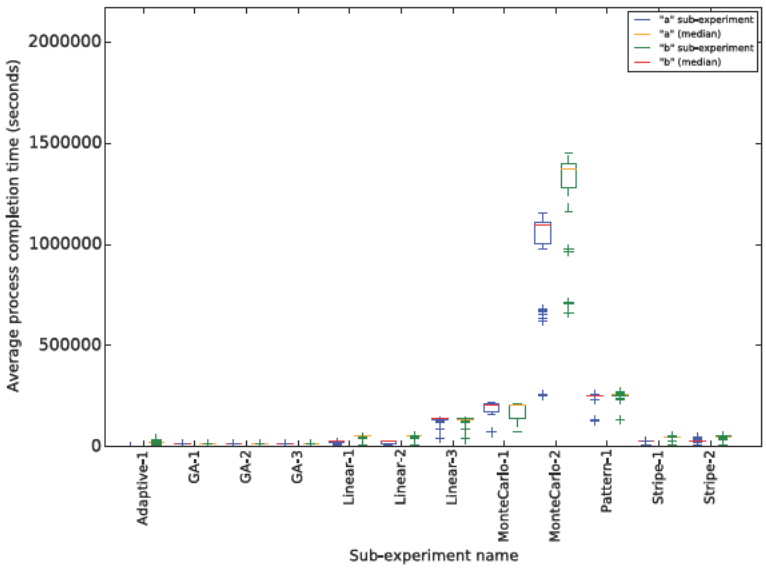


(b) Box and whisker plot depicting the total number of delivered probes recorded for all experiments. The frequency is shown across the y-axis, whist the x-axis displays the experiments. The `a` sub-experiment results are indicated by blue lines, whilst the `b` sub-experiments are denoted by green lines. Orange and red lines are included to reflect the medians of the `a` and `b` sub-experiments respectively.

Figure 4-51: Summary data of the results of the number of total delivered probes for the experiments conducted throughout the research.



(a) A summary of the average processing time for all of the experiments carried out throughout this research. The `a` sub-experiment results are indicated by blue bars, whilst the `b` sub-experiments are denoted by green bars.



(b) Box and whisker plot depicting the processing times recorded for all experiments. The processing time in seconds is shown across the y-axis, whist the x-axis displays the experiments. The `a` sub-experiment results are indicated by blue lines, whilst the `b` sub-experiments are denoted by green lines. Orange and red lines are included to reflect the medians of the `a` and `b` sub-experiments respectively.

Figure 4-52: Summary data of the results of the process completion time for the experiments conducted throughout the research.

Table 4.18: A summary of the results gathered throughout the experiments. The maximum, minimum and average values for the major dependent variables observed throughout the research (i.e. the number of successful probes, transmitted probes and processing time) are presented for each experiment.

Experiment	Successful probes			Transmitted probes			Process time (seconds)		
	Maximum	Minimum	Average	Maximum	Minimum	Average	Maximum	Minimum	Average
`Linear-1a`	0	0	0.0	4294967296	4294967296	4294967296.0	28049.6	6132.7	23758.5
`Linear-1b`	16284	16284	16284.0	4294967296	4294967296	4294967296.0	53731.6	11165.9	50441.5
`Linear-2a`	0	0	0.0	4294967296	4294967296	4294967296.0	29048.7	6440.1	21119.3
`Linear-2b`	0	0	0.0	4294967296	4294967296	4294967296.0	57160.8	11347.7	54265.8
`Linear-3a`	1	0	0.49	4294967296	4294967296	4294967296.0	139838.8	38883.6	133798.5
`Linear-3b`	16284	0	8642.09	4294967296	4294967296	4294967296.0	139353.2	38336.1	132887.3
`Stripe-1a`	0	0	0.0	4294967296	4294967296	4294967296.0	30474.3	5322.4	25543.1
`Stripe-1b`	0	0	0.0	4294967296	4294967296	4294967296.0	51639.4	10781.3	49003.8
`Stripe-2a`	0	0	0.0	4294967296	4294967296	4294967296.0	45147.4	5316.2	27753.2
`Stripe-2b`	4150	4150	4150.0	4294967296	4294967296	4294967296.0	52169.5	10721.1	50041.9
`MonteCarlo-1a`	0	0	0.0	4294967296	4294967296	4294967296.0	219874.7	71295.4	175437.9
`MonteCarlo-1b`	0	0	0.0	4294967296	4294967296	4294967296.0	215418.3	71284.3	175111.7
`MonteCarlo-2a`	4	0	1.2	4294967296	4294967296	4294967296.0	1155906.6	251762.2	1038336.4
`MonteCarlo-2b`	45	15	30.42	4294967296	4294967296	4294967296.0	1449487.2	662446.7	1318608.3
`GA-1a`	0	0	0.0	4294967296	4294967296	4294967296.0	14866.0	12368.1	13985.1
`GA-1b`	12	3	6.95	4294967296	4294967296	4294967296.0	14784.5	12283.5	13910.6
`GA-2a`	0	0	0.0	4294967296	4294967296	4294967296.0	14412.1	12353.2	13938.1
`GA-2b`	0	0	0.0	4294967296	4294967296	4294967296.0	14537.9	12288.4	13924.7
`GA-3a`	0	0	0.0	4294967296	4294967296	4294967296.0	14032.7	11838.9	13267.5
`GA-3b`	0	0	0.0	4294967296	4294967296	4294967296.0	13755.3	11805.9	13295.6
`Pattern-1a`	0	0	0.0	4284760840	4284760840	4284760840.0	261297.3	129301.7	247140.8
`Pattern-1b`	2543	2543	2543.0	4284760840	4284760840	4284760840.0	269453.6	133509.7	250607.6
`Adaptive-1a`	100	100	100.0	6553710	6553700	6553701.6	2004.6	186.2	1637.6
`Adaptive-1b`	10218	10218	10218.0	603268141	603137066	603220298.01	37656.1	2201.1	20019.7

Table 4.19: The cumulated results from all of the experiments conducted. A comparison between the dependent variables (valid probes transmitted, total probes transmitted and the process time) are provided in conjunction with the total number of simulations for each of the sub-experiment categories (`a` and `b` sub-experiments).

Experimental parameters		Experiment number	
		Randomised dataset (`a` sub-experiments)	Surveyed dataset (`b` sub-experiments)
Valid probes		(/50000)	(/41171)
	Total valid probes	10169	4187446
	Maximum	100 (0.2%)	16284 (39.55%)
	Mean	8.47 (0.02%)	3489.54 (8.476%)
	Minimum	0 (0%)	0 (0%)
	Median	0.0	8
Transmitted probes			
	Total sent probes	4724099000000	4783765000000
	Maximum	4294967296	4294967296
	Mean	3936749000.0	3986471000.0
	Minimum	6553700	603137066
	Median	4294967296	4294967296
Process time			
	Total process time	173571600	214211900
	Maximum	1155906.63	1449487.21
	Mean	144643.01	178509.88
	Minimum	186.15	2201.07
Total number of simulations			
		1200	1200

4.10 Research hypotheses

This section addresses the five primary research hypotheses that were defined in Chapter 1.

4.10.1 *H1*: Search techniques are unable to enumerate networked devices on 64 bit IPv6 subnetworks.

Hypothesis *H1* asserted that search techniques would not be able to enumerate networked devices on standard 64 bit IPv6 subnetworks. In order to test this hypothesis 24 distinct experiments were conducted against two hypothetical 64 bit IPv6 subnetworks. The null hypothesis posed no devices would be enumerated for each search operation (i.e. $H1_0 : \mu = \mu_0$). The results of the experiments disproved the null hypothesis and provided support for the alternative hypothesis that search techniques are able to enumerate networked devices on 64 bit IPv6 subnetworks (i.e. $H1_1 : \mu > \mu_0$).

The non-parametric One-Sample Wilcoxon Signed-Ranked Test with a target α value of $\alpha = 0.05$. The Wilcoxon Signed-Ranked Test is appropriate for comparing means of samples in cases where a population, or sample of a population does not conform to a normal distribution (Corder & Foreman, 2011). The null hypothesis for the test stated that the ranked means of the collected data were equal to 0. The test completed with the test statistic $V = 413,600$ and a p-value of $p < 0.001$. This indicates that in this test the null hypothesis can be rejected with a high degree of confidence and the alternate hypothesis that the ranked mean is greater than 0, can be accepted. From these results it is evident that devices are able to be enumerated on 64 bit IPv6 subnetworks using host enumeration techniques.

4.10.2 *H2*: Methods that employ random sampling do not perform better than methods that do not employ random sampling for IPv6 host enumeration.

Hypothesis *H2* asserted that search methods that employ random sampling techniques provide no performance benefit over those that do not employ random sampling for IPv6 host enumeration. Search methods that employ randomisation are commonplace in IPv4 host enumeration strategies because they provide a means to distribute probing

load across the entire address space during Internet-wide searches. Such methods had not been tested with IPv6 host enumeration prior to this research. To address this hypothesis the experimentation process tested algorithms with stochastic properties, as well as those with deterministic properties.

The results of the algorithms that employed random sampling were compared to those search algorithms that did not employ random sampling. A null hypothesis was defined for this test that there was no significant difference between the results of the stochastic algorithms against the deterministic algorithms (i.e. $H2_0 : \mu = \mu_0$). The alternative hypothesis indicated that there was a difference between the data in each level, and that the deterministic algorithms performed greater than the stochastic algorithms (i.e. $H2_1 : \mu > \mu_0$).

The experiments that involved the use of random sampling search techniques, or whose operations were unpredictable, and therefore were categorised as stochastic, as well as those that involved the use of deterministic search techniques and therefore were categorised as deterministic, are included in Table 4.20 (extracted from Chapter 3).

Table 4.20: Table of the algorithms designed for the study and their classification as stochastic or deterministic in nature.

Stochastic	Deterministic
MonteCarlo	Linear
GA	Stripe
Adaptive	Pattern

Table 4.21 illustrates the results of the deterministic algorithms compared to the stochastic algorithms for the randomly generated dataset. Similarly, Table 4.22 illustrates the results of the deterministic algorithms compared to the stochastic algorithms for the surveyed dataset. Finally, Table 4.23 highlights the comparison between all of the sub-experiments that utilised random sampling (i.e. the stochastic search algorithms) against those that did not (i.e. the deterministic search algorithms) for both the `a` and `b` sub-experiments.

In order to determine whether the differences between the populations is statistically significant inferential statistics were applied to:

- The results of the `a` sub-experiments.

- The results of the `b` sub-experiments.
- The combined results of both sub-experiments (`a` and `b`).

The results were unremarkable from either the stochastic or deterministic categories when applied to the randomly generated dataset, as is evident in Table 4.21. A visual comparison of means for the `a` sub-experiments indicates that a difference between the stochastic and deterministic search algorithms is not apparent. To determine whether there was a statistically significant difference between the two groups, the one-tailed Wilcoxon Rank-Sum test was applied. The Wilcoxon Rank-Sum test is appropriate for non-parametric hypothesis testing involving comparing two independent variables (Corder & Foreman, 2011). This test compared the distribution of differences between the two groups, and determined whether a difference existed at the 95% confidence interval. The Wilcoxon test passed with a test statistic $W = 139,790$ and a p-value of $p = 1$. This indicates that there is no significant difference between the stochastic search algorithms and the deterministic search algorithms where the algorithms were applied to the randomly generated dataset (i.e. the `a` sub-experiments). With this result, there is sufficient evidence to accept the null hypothesis that the results of the deterministic group are not significantly greater than the results of the stochastic group (i.e. that $\mu \leq \mu_0$).

Contrarily, the comparison of the differences for the `b` experiments indicates that a potential difference between the results of the stochastic search algorithms and the deterministic search algorithms exists. Again, to confirm whether this difference was statistically significant at the 95% confidence interval, the one-tailed Wilcoxon Rank-Sum test was applied. The Wilcoxon test passed with a test statistic of $W = 236,740$ and a p-value of $p < 0.001$ highlighting that the results of the deterministic search algorithms are significantly greater than those of the stochastic search algorithms applied to the surveyed dataset (i.e. that $\mu > \mu_0$). It is possible to confidently reject the null hypothesis in this instance and conclude that the deterministic search algorithms performed better than the stochastic search algorithms when applied to the surveyed dataset.

Finally the combined results of all of the deterministic search algorithms were com-

Table 4.21: A comparison of means of successful probes, between algorithms that employed random sampling and deterministic algorithms tested against the randomised dataset. The Wilcoxon Rank-Sum test statistic and p-Value indicate a significant difference between the populations does not exist.

Algorithm type	Experiment number	Successful probes
Deterministic		
	Linear-1a	0.00
	Linear-2a	0.00
	Linear-3a	0.49
	Pattern-1a	0.00
	Stripe-1a	0.00
	Stripe-2a	0.00
Stochastic		
	Adaptive-1a	100.00
	GA-1a	0.00
	GA-2a	0.00
	GA-3a	0.00
	MonteCarlo-1a	0.00
	MonteCarlo-2a	1.20
Test statistic		**139790**
p-Value		**1**

Table 4.22: A comparison of means of successful probes, between algorithms that employed random sampling and deterministic algorithms tested against the surveyed dataset. The Wilcoxon Rank-Sum test statistic and p-Value indicate a significant difference between the populations is present.

Algorithm type	Experiment number	Successful probes
Deterministic		
	Linear-1b	16284.00
	Linear-2b	0.00
	Linear-3b	8642.09
	Pattern-1b	2543.00
	Stripe-1b	0.00
	Stripe-2b	4150.00
Stochastic		
	Adaptive-1b	10218.00
	GA-1b	6.95
	GA-2b	0.00
	GA-3b	0.00
	MonteCarlo-1b	0.00
	MonteCarlo-2b	30.42
Test statistic		**236740**
p-Value		**<0.001**

pared against the results of the stochastic search algorithms to determine if a significant difference existed between the groups. Again, the one-tailed Wilcoxon Rank-Sum test

was employed to test the null hypothesis that $\mu = \mu_0$, and to provide support for the alternative hypothesis that $\mu > \mu_0$ and that the difference was significant at the the 95% confidence interval. The Wilcoxon test passed with a test statistic of $W = 748,720$ and a p-value of $p = 0.026$ highlighting that the results of the deterministic search algorithms are significantly greater than those of the stochastic search algorithms when applied to the combined results of the randomised and surveyed datasets (i.e. that $\mu > \mu_0$). It is possible to confidently reject the null hypothesis in this instance and conclude that the deterministic search algorithms performed better across all of the experiments that were carried out.

Table 4.23: A comparison of means of successful probes between algorithms that employed random sampling, against those that employed deterministic algorithms across both datasets. The Wilcoxon Rank-Sum test statistic and p-Value indicate a significant difference between the populations is present.

Algorithm type	Experiment number	Successful probes
Deterministic		
	Linear-1a	0.00
	Linear-1b	16284.00
	Linear-2a	0.00
	Linear-2b	0.00
	Linear-3a	0.49
	Linear-3b	8642.09
	Pattern-1a	0.00
	Pattern-1b	2543.00
	Stripe-1a	0.00
	Stripe-1b	0.00
	Stripe-2a	0.00
	Stripe-2b	4150.00
Stochastic		
	Adaptive-1a	100.00
	Adaptive-1b	10218.00
	GA-1a	0.00
	GA-1b	6.95
	GA-2a	0.00
	GA-2b	0.00
	GA-3a	0.00
	GA-3b	0.00
	MonteCarlo-1a	0.00
	MonteCarlo-1b	0.00
	MonteCarlo-2a	1.20
	MonteCarlo-2b	30.42
Test statistic		**748720**
p-Value		**0.026**

Overall the null hypothesis for Hypothesis *H2* been accepted, and it can be concluded that stochastic search methods are inappropriate for off-link host enumeration against 64 bit subnetworks.

4.10.3 *H3*: Randomly generated interface identifiers do not affect the performance of IPv6 host enumeration search algorithms.

Hypothesis *H3* asserted that networks consisting of randomly generated IPv6 IIDs would affect the performance of IPv6 host enumeration search algorithms. To address this hypothesis, a comparison between the results of the `a` and `b` sub-experiments has been conducted. As mentioned in Chapter 3 the `a` sub-experiments applied the subject algorithm (the independent variable) against the dataset that was constructed using pseudorandom methods. This dataset was constructed to replicate an IPv6 network that used stochastic address generation strategies, such as the SLAAC with privacy extensions, HBAs, or CGA-based IID construction schemes. Similarly, the `b` experiments tested the subject algorithm against the dataset that was gathered from real-world survey data. This dataset was used to replicate a hypothetical network that had been configured with a variety of address types, including incremental assignment, modified EUI-64-based SLAAC, randomised allocation, and pattern-based configuration.

For each of the experiments conducted, the number of successful probes of all of the sub-experiments were compared. These comparisons were conducted using an Wilcoxon Rank-Sum test, to test the $H3_0$ null hypothesis that the ranked sums were the same for each sub-experiment (i.e. that $H3_0 : \mu = \mu_0$). The results of the Wilcoxon Rank-Sum test, highlighting the significance at the confidence interval of 95% (i.e. that $p < 0.05$) for each experiment are included in the corresponding experiment's results section above, and are summarised in Table 4.24 below.

It was observed that out of the 12 major experiments conducted, only five failed to display a significant difference between the ranked sums of the `a` and `b` sub-experiments. From this, an overall comparison of the successful probes for both groups (i.e. sub-experiments `a` and `b`) across all of the experiments was compiled and tested. This was performed to determine whether a significant difference exists between the two groups.

Table 4.24: Results of hypothesis testing between the `a` and `b` sub-experiment populations for each experiment at the 95% confidence interval (i.e. $\alpha = 0.05$). The test statistic for the Wilcoxon Rank-Sum test is included, along with the corresponding p-value for the test.

Experiment number	Test statistic	p-Value
Linear-1	0	p<0001
Linear-2	2525	1
Linear-3	170	p<0001
Stripe-1	2525	1
Stripe-2	0	p<0001
MonteCarlo-1	2525	1
MonteCarlo-2	0	p<0001
GA-1	0	p<0001
GA-2	2525	1
GA-3	2525	1
Pattern-1	0	p<0001
Adaptive-1	0	p<0001

Again, the Wilcoxon Rank-Sum test was chosen to test the null hypothesis that there is no difference at the confidence interval of 95% (i.e. that $p < 0.05$) between the ranked sums of sub-experiments `a` and sub-experiments `b` across all of the experiments. The results of this test concluded with a p-value of $p < 0.001$ and a test statistic of $W = 387541.0$, indicating that the null hypothesis can be rejected and that a significant difference exists between the ranked sums of both groups. From the results it is clearly evident that the sub-experiments that targeted the network of stochastically generated addresses performed worse than those that targeted the network that was configured with real-world addresses.

The null hypothesis has therefore been rejected in this case. There was a significant difference identified between the two IPv6 address datasets. The experiments that searched the randomly allocated IPv6 address dataset performed significantly worse those that searched the surveyed dataset. This indicates that randomly generated addresses do negatively impact on the performance of IPv6 host enumeration search algorithms.

4.10.4 *H4*: Search methods that employ machine learning are unable to enumerate networked devices on 64 bit IPv6 subnetworks.

Hypothesis *H4* was concerned with identifying whether machine learning algorithms and methods could be used to enumerate devices on IPv6 subnetworks. The null hypothesis posed that machine learning methods are unable to enumerate any devices (i.e. $H4_0 : \mu = \mu_0$). The results of the experiments were used to disprove the null hypothesis and provide support for the alternate hypothesis that machine learning search techniques are able to enumerate networked devices on 64 bit IPv6 subnetworks (i.e. $H4_1 : \mu > \mu_0$). In order to disprove the null hypothesis, the results from the experiments that employed machine learning strategies, were compared against an expected result of 0. Table 4.25 displays the search algorithms that were classified as machine learning-based, and those that were not.

Table 4.25: Table of the algorithms designed for the study and their classification indicating whether machine learning methods are employed or not.

Machine learning	Non-machine learning
GA	Linear
Adaptive	Stripe
	MonteCarlo
	Pattern

The non-parametric One-Sample Wilcoxon Signed-Ranked Test was used to test the hypothesis with a target α value of 0.05. The null hypothesis for the test stated that the ranked means of the collected data were equal to 0. The test completed with the test statistic $V = 45,150$ and a p-value of $p < 0.001$. This indicates that in this test the null hypothesis can be rejected with a high degree of confidence and the alternate hypothesis that the ranked mean is greater than 0, can be accepted. Based upon these results, it can be concluded that machine learning techniques can be used to enumerate device on IPv6 networks.

4.10.5 *H5*: Search methods that employ machine learning do not perform better than search methods that do not employ machine learning for IPv6 host enumeration.

Hypothesis *H5* was concerned with identifying whether machine learning algorithms and methods could perform better than non-machine learning methods for IPv6 host enumeration. In this context a higher number of successful probes indicates better performance. The null hypothesis posed that machine learning methods do not perform better than non-machine learning techniques for the task. In order to disprove the null hypothesis, the results from the experiments that employed machine learning strategies, were compared against those that didn't.

Table 4.26 highlights the results of the machine learning algorithms compared against the non-machine learning algorithms for the randomly generated dataset (i.e. the `a` sub-experiments). Similarly, Table 4.27 displays the results of the machine learning algorithms compared against the non-machine learning algorithms for the surveyed dataset (i.e. the `b` sub-experiments). Finally, Table 4.28 reveals the results of the machine learning algorithms compared against the non-machine learning algorithms across all of the sub-experiments. In order to test the hypothesis and determine whether machine learning algorithms perform better than non-machine learning algorithms when applied to IPv6 host enumeration, comparisons were made between:

- The results of the `a` sub-experiments.
- The results of the `b` sub-experiments.
- The combined results of both sub-experiments (`a` and `b`).

A visual observations of the groups for this test (available in Table 4.26) indicates that a difference between the two populations is present. In order to determine whether the results between the identified groups differed significantly at the 95% confidence interval (i.e. $p < 0.05$), the Wilcoxon Rank-Sum test was used. When applied to the `a` sub-experiments, the Wilcoxon test passed with a test statistic of $W = 181850$ and a p-value of $p < 0.001$ indicating that a significant difference between the two populations exists. The results for this test are included in Table 4.26.

Comparing the results from the sub-experiments performed against the surveyed dataset between the machine learning and non-machine learning categories reveals no visible difference between the groups. To determine whether a statistically significant difference between the data exists at the 95% confidence interval ($p < 0.05$), the Wilcoxon Rank-Sum test was employed. This test also failed with a test statistic of $W = 129270$ and a p-value of $p = 1$ indicating that the results of the machine learning experiments are not significantly greater than the results of the non-machine learning experiments.

Finally, the results of all of the sub-experiments were tested. The data that were gathered from the sub-experiments that employed machine learning techniques, compared to those that did not employ machine learning techniques were tested to determine whether a statistically significant difference existed between the groups. Again, the Wilcoxon Rank-Sum test was used to determine whether the ranked sums of the two groups were different at the 95% confidence interval (i.e. $p < 0.05$). The outcome of this test again failed to highlight a significant difference between the two populations, with a test value of $W = 629410$ and a p-value of $p = 0.776$.

The tests performed against the collected data failed to reveal a significant difference between the results of the two independent groups; the algorithms that employed machine learning and those that did not employ machine learning. From the results of testing the hypothesis *H5* the null hypothesis must be accepted. It can therefore be concluded that the algorithms that employed machine learning did not provide any performance benefit over the algorithms that did not employ machine learning for IPv6 host enumeration.

Table 4.26: A comparison of means of successful probes, and the successful probes to process time ratio, between experiments that employed machine learning and non-machine learning algorithms tested against the randomised dataset. The outcome of the Wilcoxon Rank-Sum test is included, and highlights a significant difference between the two populations at the 95% confidence interval.

Algorithm type	Experiment number	Successful probes
Machine learning		
	Adaptive-1a	100.00
	GA-1a	0.00
	GA-2a	0.00
	GA-3a	0.00
Non-Machine learning		
	Linear-1a	0.00
	Linear-2a	0.00
	Linear-3a	0.49
	Pattern-1a	0.00
	Stripe-1a	0.00
	Stripe-2a	0.00
	MonteCarlo-1a	0.00
	MonteCarlo-2a	1.20
Test statistic		**181850**
p-Value		**<0.001**

Table 4.27: A comparison of means of successful probes, and the successful probes to process time ratio, between experiments that employed machine learning and non-machine learning algorithms tested against the surveyed dataset. The outcome of the Wilcoxon Rank-Sum test is included, and highlights a lack of a significant difference between the two populations at the 95% confidence interval.

Algorithm type	Experiment number	Successful probes
Machine learning		
	Adaptive-1b	10218.00
	GA-1b	6.95
	GA-2b	0.00
	GA-3b	0.00
Non-Machine learning		
	Linear-1b	16284.00
	Linear-2b	0.00
	Linear-3b	8642.09
	Pattern-1b	2543.00
	Stripe-1b	0.00
	Stripe-2b	4150.00
	MonteCarlo-1b	0.00
	MonteCarlo-2b	30.42
Test statistic		**129270**
p-Value		**1**

Table 4.28: A comparison of means of successful probes, and the successful probes to process time ratio, between experiments that employed machine learning and non-machine learning algorithms against the results for both datasets. The outcome of the Wilcoxon Rank-Sum test is included, and highlights a lack of a significant difference between the two populations at the 95% confidence interval.

Algorithm type	Experiment number	Successful probes
Machine learning		
	Adaptive-1a	100.00
	Adaptive-1b	10218.00
	GA-1a	0.00
	GA-1b	6.95
	GA-2a	0.00
	GA-2b	0.00
	GA-3a	0.00
	GA-3b	0.00
Non-Machine learning		
	Linear-1b	16284.00
	Linear-1a	0.00
	Linear-2b	0.00
	Linear-2a	0.00
	Linear-3b	8642.09
	Linear-3a	0.49
	Pattern-1b	2543.00
	Pattern-1a	0.00
	Stripe-1b	0.00
	Stripe-1a	0.00
	Stripe-2b	4150.00
	Stripe-2a	0.00
	MonteCarlo-1a	0.00
	MonteCarlo-1b	0.00
	MonteCarlo-2a	1.20
	MonteCarlo-2b	30.42
Test statistic		**629410**
p-Value		**0.776**

Chapter 5

Discussion of Findings

This chapter explores the relationship between the results of the experiments carried out in this study, the research hypotheses and the research questions. The primary research hypotheses were first introduced in Section 1.4. The experiments described in Chapter 3 were used to test the research hypotheses. The outcomes of the hypothesis testing from Chapter 4 served to answer the research questions described in Section 1.3. The implications of how these results shape the landscape about the research topic are also discussed in this chapter.

5.1 Outcomes of the research questions

The primary research questions that guided the research project were:

RQ1 *Can networking devices be enumerated on IPv6 64 bit subnetworks using host enumeration techniques?*

RQ2 *Are stochastic searching methods more efficient than deterministic searching methods when enumerating IPv6 hosts within a single 64 bit subnetwork?*

RQ3 *Do stochastic address allocation schemes within a single 64 bit subnetwork inhibit IPv6 host enumeration strategies?*

RQ4 *Can machine-learning search methods be used to enumerate devices on a 64 bit IPv6 subnetwork?*

RQ5 Are machine-learning search methods more efficient than non-machine learning based methods when enumerating IPv6 hosts within a single 64 bit subnetwork?

After testing the primary hypotheses of the research using the experiments outlined in Chapter 3, the results were used to provide answers to the research questions. The relationship between the research questions and the hypotheses is displayed in Table 5.1

Table 5.1: Table displaying the relationship between the primary research questions and the hypotheses.

Research question	Corresponding hypothesis
RQ1: Can networking devices be enumerated on IPv6 64 bit subnetworks using host enumeration techniques in a timely manner?	H1: Search techniques are unable to enumerate networked devices on 64 bit IPv6 subnetworks.
RQ2: Are stochastic searching methods more effective than deterministic searching methods when enumerating IPv6 hosts within a single 64 bit subnetwork?	H2: Methods that employ random sampling do not perform better than methods that do not employ random sampling for IPv6 host enumeration.
RQ3: Do stochastic address allocation schemes within a single 64 bit subnetwork inhibit IPv6 host enumeration strategies?	H3: Randomly generated interface identifiers do not affect the performance of IPv6 host enumeration search algorithms.
RQ4: Can machine learning search methods be used to enumerate devices on a 64 bit IPv6 subnetwork?	H4: Search methods that employ machine learning are unable to enumerate networked devices on 64 bit IPv6 subnetworks.
RQ5: Are machine learning search methods more efficient than non-machine learning based methods when enumerating IPv6 hosts within a single 64 bit subnetwork?	H5: Search methods that employ machine learning do not perform better than search methods that do not employ machine learning for IPv6 host enumeration.

5.1.1 *RQ1*: Can networking devices be enumerated on 64 bit IPv6 subnetworks using host enumeration techniques?

The literature surrounding this topic, as discussed previously in Section 2.3, supports the notion that IPv6's address space is too vast to enumerate efficiently. Under RFC-4291 (Hinden & Deering, 2006), a standard IPv6 subnetwork size is defined as being 64 bits long. This translates to over 18 quintillion unique addresses per subnet. At a rate of 1 million packets per second, would take 584,942.42 years to exhaustively enumerate a 64 bit subnetwork. Hypothesis *H1* sought to test the notion that IPv6 networks are not susceptible to host enumeration, and asserted that networked devices on 64 bit IPv6 subnetworks are unable to be enumerated using search techniques. To test this hypothesis the combined results of all 24 sub-experiments were considered in *H1*.

From the results of the experiments conducted, the null hypothesis for Hypothesis

H1 was rejected at the 95% confidence interval (i.e. $\alpha = 0.05$), indicating that it is possible to enumerate devices on an IPv6 subnetwork. In particular, throughout this research it was observed that a mean of 1749 successful probes were transmitted per sub-experiment. These results represents 3.5% of the total number of configured nodes for the randomly generated dataset, and 4.25% of the total number of configured nodes for the surveyed dataset. Considering the scope of the problem, these results are remarkable, and represent a significant finding for the research. Furthermore, the maximum number of successful probes discovered during a single simulation was 16,284 during the `Linear-1b` sub-experiments which represented 39.55% of the total configured hosts on the surveyed dataset. The outcomes of the hypothesis testing highlight that host enumeration is possible against off-link 64 bit IPv6 subnetworks. This finding provides a firm foundation for the expansion of further research into potential IPv6 host enumeration search techniques and methods, and contradicts the accepted position that IPv6 host enumeration is futile.

According to Jara, Ladid and Skarmeta (2013) and Zanella, Bui, Castellani, Vangelista and Zorzi (2014), IPv6 is positioned to see significant usage in IoT systems. IoT systems typically include a large number of network-connected sensors (in excess of 300, according to Zanella et al. (2014)), making IPv6 an appropriate choice for network level communications. Off-link host enumeration has an impact on the security of the connected devices in IoT systems on IPv6 networks. The ability to locate nodes on a network exposes nodes to potential attack even with an address space as vast as a standard IPv6 subnetwork. It has been observed that IPv6 devices are not entirely secure from off-link host enumeration. Although the large address space does afford devices more protection than equivalent IPv4 networks from host enumeration, the means by which addresses are constructed can reduce the protection significantly.

5.1.2 *RQ2*: Are stochastic searching methods more efficient than deterministic searching methods when enumerating IPv6 hosts within a single 64 bit subnetworks?

The literature review process revealed two major categories of search algorithms used for conducting host enumeration in IPv4; using linear search to sequentially enumerate the target address space, or using some randomisation function to enumerate the target address space in a non-sequential fashion. To determine whether the approaches can translate effectively to the IPv6 address space, the outcomes of Hypothesis *H2* must be considered.

Hypothesis *H2* suggested that search techniques that utilise random sampling do not perform better than those that don't. The null hypothesis for Hypothesis *H3* was accepted in light of the results of this research. That is to say that the results strongly suggest that random sampling approaches are not appropriate for searching 64 bit IPv6 subnetworks. It was observed that the majority of the algorithms that depended on random sampling failed to produce significant or meaningful results. In the majority of the experiments, the purely stochastic methods failed to yield a single successful probe throughout all simulations. The exception to this was the Adaptive-heuristic search algorithm, which successfully managed to enumerate an average of 10,218 hosts from the surveyed dataset.

The results of the sub-experiments involving the linear search algorithms against the surveyed IPv6 dataset validate that host enumeration strategies used in the IPv4 realm are valid strategies to use against IPv6 networks. Out of all of the experiments conducted, the linear search algorithm performed the strongest on average. This is evidenced by the results of `Linear-1b` and `Linear-3b`, which successfully enumerated an average of 16,284 and 8,642 nodes respectively. These results show that the linear search strategy is still a valid enumeration strategy against IPv6 networks. The time constraints of conducting an exhaustive search, however, need to be acknowledged. As is suggested by the literature and supported by the results of this research, with current computing resources, it would certainly be infeasible to conduct an exhaustive search against an IPv6 network. Not only would it be infeasible with respect to the time

costs associated with the exercise, an exhaustive search of the address space might not reveal all of the hosts on the network. Hosts on a computer network may connect and disconnect many times throughout the lifespan of a search operation.

These findings rule the Monte Carlo method out from being a candidate for IPv6 host enumeration strategies. Other strategies, such as using LCGs or generalised Feistel algorithms (such as those employed by `zmap` and `masscan`) would not be recommended approaches to enumerating an off-link, 64 bit IPv6 network. Instead it would be recommended to focus on deterministic search methods (such as the Linear search or Stripe search algorithms), or search techniques that exploit predictability or repeating patterns in IPv6 addresses (such as the Pattern-based heuristic search algorithm). Despite being classified as stochastic, the adaptive-heuristic search method performed particularly well and is a recommended candidate algorithm for usage in host enumeration exercises.

Stochastic search methods have been used in IPv4 host enumeration strategies as a mechanism to randomly sample the address space, as well as to distribute the load of searching. The density of public IPv4 address allocation and usage enable random sampling search methods to operate effectively, since the majority of the relatively small address space is utilised. The same is not true for IPv6. The public IPv6 address space is sparsely populated when compared to IPv4's public address space. In relation to IPv6 search efforts, this research has shown random sampling to be ineffective. The problem can be likened to the "needle in a haystack" problem, except on a much larger scale.

5.1.3 *RQ3*: Do stochastic address allocation schemes within a single 64 bit subnetworks inhibit IPv6 host enumeration strategies?

Stochastically generated IIDs are a proposed solution to address the privacy issues that have been identified with IPv6 address construction methods. In particular, permanent or semi-permanent IIDs can be used to track devices even as they roam amongst different networks. Additionally, with SLAAC generated addresses that conform to the modified EUI-64 construction scheme, the link layer address of a device is derivable from

the IPv6 address. Currently privacy extensions for SLAAC are recommended to mitigate the risk of exposing potentially identifiable information (in particular a device's MAC address), to the wider public Internet. Other methods of generating randomised addresses, such as hash-based addresses, or cryptographically generated addresses, have also been proposed to address the privacy concerns arising from deterministic address construction schemes. Hypothesis *H3* was concerned with determining whether these proposed stochastically generated IIDs provide protection against host enumeration in IPv6 networks.

As was expected, in light of the literature, the research failed to disprove the null hypothesis of Hypothesis *H3*. The majority of experiments that targeted the network configured with randomly generated addresses failed to yield any significant results. The maximum number of hosts discovered on the network configured with randomly generated addresses was 100, or 0.2% of the total configured devices on that network. This is in contrast to 39.55% of devices that were discovered on the network that was configured to reflect real-world addresses, which represents a significant portion of the configured addresses. It can be concluded, therefore, that stochastic address allocations schemes do inhibit off-link IPv6 host enumeration strategies.

The observations made from the research align with the available literature that suggest that it would be difficult to locate addresses that use randomisation functions to generate IIDs. Since the generation of secure addresses can be likened to a cryptographic problem, cryptographic approaches are required to resolve it. The approaches used with HBAs and CGAs are therefore appropriate. That is provided that the randomisation function is correctly implemented, and in the case of PRNG-based solutions, seeded appropriately. This caveat is to reduce the opportunity for patterns, or other deterministic features to arise from the address construction process, which would reduce the security of the approach. This finding does reinforce the notion that good privacy and protection from off-link host enumeration efforts can be achieved by using stochastic means to generate IIDs. However, randomised address construction should not be the only defence measure employed. Such an approach should be taken as a part of a complete defence in depth strategy.

5.1.4 *RQ4*: Can machine learning search methods be used to enumerate devices on a 64 bit IPv6 subnetwork?

Machine learning systems have thus far not been applied to the problem of host enumeration, let alone specifically against IPv6 systems. This research has pioneered the approach. The adaptive and GA algorithms used machine learning in order to influence their search patterns, whilst the other algorithms tested did not. The adaptive search algorithm employed a decision making system in order to critically assess each discovered address and alter its search approach accordingly. Discovered addresses were classified, and then further targets were added for searching based upon the outcome of the classification.

The GA search algorithm used an evolutionary process to 'breed' potential search candidates. Target addresses were subject to fitness testing to determine whether they were strong candidates for further breeding. This strategy was chosen since it was expected that the algorithm would probe addresses in clusters throughout the address space. It was shown through the various pilot studies for the algorithm that there was a tendency for the GA to group targets throughout the address space. Like the adaptive search algorithm, the GA represents a novel approach to host enumeration that had not been attempted prior to this study.

To determine whether machine learning search methods could be used to enumerate devices on IPv6 subnetworks, Hypothesis *H4* was devised and tested. Hypothesis *H4* asserted that machine learning methods cannot be used to enumerate devices on IPv6 subnetworks. The test revealed that at the 95% confidence interval, the results of the machine learning experiments were significantly greater than 0. Consequently, the null hypothesis was rejected, and the alternative hypothesis that machine learning search techniques could be used to enumerate devices on 64 bit IPv6 subnetworks was accepted.

The GA performed poorly overall, successfully probing a maximum of 0 out of 50,000 nodes against the randomised dataset and a maximum of 12 out of 41,171 possible nodes to probe against the surveyed dataset (0.03% of the possible hosts on the network). By contrast, the adaptive algorithm performed better than the majority

of the algorithms tested, ranking second overall by average successful probes. The algorithm successfully enumerated on average 10,218 hosts on the surveyed network (or 24.8% of possible hosts on that network), and 100 hosts on the randomly generated dataset (or 0.2% of possible hosts on that network).

It was observed from this research that machine learning algorithms can be used to enumerate hosts on an IPv6 network. Machine learning approaches have been successfully applied to a number of computer science problems. The results of this research highlight that machine learning techniques should be considered for further research into emerging host enumeration algorithms.

5.1.5 *RQ5*: Are machine learning searching methods more efficient than non-machine learning based methods when enumerating IPv6 hosts within a single 64 bit subnetworks?

To answer this research question, Hypothesis *H5* was tested. Hypothesis *H5* asserted that machine learning methods were not more effective than non-machine learning methods for IPv6 host enumeration purposes. The null hypothesis for Hypothesis *H5* was accepted since there was no statistically significant difference between the algorithms that employed machine learning and those that did not employ machine learning at the 95% confidence interval.

Although a lack of statistical difference indicates that machine learning approaches did not perform better than non-machine learning approaches during this study, there is still potential for machine learning in IPv6 host enumeration. The adaptive search algorithm, for example, successfully enumerated an average of 10,218 hosts across all simulations conducted across the surveyed dataset. It also successfully enumerated an average of 100 hosts against the randomly generated dataset. These results ranked amongst the highest recorded results throughout the research.

The machine learning domain shows promise for host enumeration, however has not been fully explored by this research. Further development into machine learning approaches for host enumeration is required to improve the performance of the algorithms.

5.2 Implications of research

This research has made a number of significant contributions to knowledge. These contributions are noted in the coming sections.

5.2.1 Host enumeration classification framework

A classification system for host enumeration strategies was introduced in Chapter 2. Two major categories of host enumeration were identified: active and passive enumeration. Active enumeration involves strategies that involve the direct probing of systems in order to enumerate devices. Examples of active enumeration include TCP SYN scanning, ICMP ping sweeps, or ARP ping sweeps. Within the active enumeration category, a five component framework for active host enumeration methods was introduced. The five major components identified for active host enumeration methods were:

- The target address space: the address(es) or range of addresses that are to be probed during the host enumeration exercise. This forms the object of interest for the host enumeration.
- Probe targets: The delivery addresses of the probes. Usually the same as the target address space, but in certain situations may differ (e.g. when querying IPv6 local multicast groups to enumerate off-link nodes, or when targeting link layer addresses when using ARP ping to enumerate devices).
- The search algorithm: the order by which targets are probed. Typically the search algorithm is either sequential or uses some randomisation function.
- The protocol: the protocol that is used to probe the target address(es) and measure the response. For example TCP SYN segments, ICMPv6 echo requests, or ARP requests.
- The probe payload: the data attached to the probe. In some cases, for vulnerability testing or malicious activities, payload may be included in the probe to trigger a response on a target. As an example, the Shellshock vulnerability in

2014 could be exploited by sending malformed Bash strings as the payload (in particular the User Agent string) in HTTP requests. In other cases the payload may simply be arbitrary data.

Passive enumeration involved conducting host enumeration without directly probing the target system (i.e. the reconnaissance subject). This can be achieved through strategies such as passive network monitoring, querying ancillary network services (e.g. DNS, SNMP, etc.) or looking at DHCP leases or ARP/NDP tables. Within the passive enumeration category four major components were identified:

1. The reconnaissance subject: The target of interest to the enumeration exercise. This could be network, website, domain name, host address, etc.

2. The reconnaissance object: The system that is under observations for determining live hosts. For example, in a DNS enumeration the DNS server being queried would be the reconnaissance object.

3. The protocol: The protocol refers to the underlying protocol being used against the reconnaissance object to enumerate devices. Continuing with the DNS enumeration example, the protocol would be the DNS protocol.

4. The payload: The information that is being examined to gain information about the reconnaissance subject (such as enumeration of devices). For example, the DNS requests and responses containing valid IP addresses.

5.2.2 Privacy issues with IPv6 address construction

This research highlighted potential privacy issues with IPv6 address construction that could be used to monitor activities of device owners. The research identified that three classes of address construction could be inferred without any context or prior knowledge of how an address was formed. These classes were;

- EUI-64 addresses (or SLAAC addresses without privacy extensions) which included IIDs that had undergone the modified EUI-64 process to convert a 48 bit link-layer MAC address into a 64 bit IPv6 IID.

- Incremental or statically assigned addresses, which included addresses that were low range numbers, or included hexadecimal word substitution, or conformed to an assignment policy specified by an administrator.
- Randomised addresses (or SLAAC with privacy extensions), which included addresses that have been cryptographically generated or are indistinguishably random.

These findings were published in Carpene and Woodward (2012) and the identified classes formed the basis for the classification system that was developed for this research.

5.2.3 IPv6 address classification system

A classification system developed during the course of this study (which has been published in Carpene et al. (IN PRESS)) provides evidence that machine learning systems, such as ANNs and naive Bayesian classifiers, can be used to classify IPv6 address IID construction. This classification system utilised machine learning algorithms to determine whether IPv6 IIDs were constructed using modified EUI-64, a randomisation function, or whether they had be assigned using an incremental or static addressing scheme.

Such a classification system has potential benefits for usage, not only in IPv6 host enumeration search strategies, but with other systems that process IP addresses, such as IDS, IPS and security information and event management (SIEM) systems. IPv6 address classification could be used to provide metrics for detection and classification of threat actors for alerted activity in threat detection systems.

5.2.4 IPv6 usage survey

The first phase of the research (Phase 1: Perform survey of IPv6 usage) involved a passive host enumeration survey of public DNS services to gather IPv6 addresses. It was revealed that despite recommendations and best practices from authorities such as the IETF, the majority of collected addresses from public information sources conformed to the incremental or statically assigned address class. This exercise represented novel

research that shed light on the real-world usage habits of IPv6. The IPv6 survey research was published in Carpene et al. (IN PRESS).

5.2.5 Significant advancements to off-link IPv6 host enumeration

This research marks the first empirical study conducted that tests the efficacy of various host enumeration algorithms against IPv6 networks. The literature surrounding the topic suggests that the address space is too vast to feasibly enumerate. However, whilst tools have been developed for the task this marks the first formal research conducted about the topic.

The effectiveness of multiple host enumeration algorithms were compared to determine which algorithms are candidates for host enumeration exercises against IPv6 networks. Algorithms that are currently employed during IPv4 host enumeration exercises were tested for their efficacy in IPv6 host enumeration. An empirical comparison and analysis of host enumeration search methods has thus far not been conducted, and constitutes a significant contribution to knowledge from this research. This analysis revealed that some search methods are not suited to IPv6 host enumeration. In particular, stochastic, or randomised search algorithms failed to enumerate any meaningful quantity of hosts during the experiments. The deterministic algorithms performed substantially better overall, implying that efforts should be directed towards them for further development of search algorithms.

A number of novel approaches to searching a 64 bit IPv6 subnetwork were presented in this research project. The adaptive heuristic host enumeration algorithm represents a novel approach to host enumeration and a significant contribution to knowledge made by this research. The algorithm incorporates both active and passive enumeration strategies, as well as machine learning to improve the probability of identifying hosts. This approach has, thus far, not been attempted as a strategy for enumeration. The algorithm (described in Chapter 3) first conducts passive enumeration of the target address space. The addresses that are gathered from the passive enumeration are classified using a machine learning classification system, to determine how the address IIDs were constructed. Based upon the classification results of each address, new targets are computed, and compiled into the target address list, for searching.

Additionally, the use of machine learning and decision making constructs for host enumeration is a novel approach to the problem that was tested during this research. As mentioned above, the adaptive heuristic search algorithm employed machine learning in order to classify addresses, as well as to influence its search operations. A genetic algorithm was also tested that employed machine learning. The genetic algorithm used an evolutionary process to generate target addresses and probe them. It was discovered that machine learning systems can be effective for use in host enumeration. This foundational research paves the way for future studies to expand upon, and improve the applicability of machine learning search algorithms for off-link IPv6 host enumeration.

5.2.6 Host enumeration scenarios

In addition to the contributions to knowledge detailed above, the following real-world applications of the research findings are presented. The findings of this research influence the approach that should be taken when applying host enumeration to real-world IPv6 networks. The findings have identified recommended approaches for host enumeration, depending on the information that is available to the initiator of the enumeration exercise.

5.2.6.1 Blind search

In real-world scenarios actors would have access to varying amounts of information when performing host enumeration. For example, an attacker targeting a public network with disclosed public services may have access to information such as DNS records for nodes. On the contrary, a network administrator on an entirely private network may have no additional information available other than the protocol (e.g. IPv6 or IPv4) that the network uses. Blind searching refers to situations where additional context or information about the target is not available, effectively leaving the agent blind.

In a situation where an actor has no prior knowledge about the target IPv6 network, nor any prior information about common protocol usage (such as those tested in stochastic settings), the linear search provides the best chance for locating networked nodes. However, if the nodes in the search space are randomly distributed, the results of this study show that there is effectively no chance of locating such addresses. At this

point, the actor should consider alternative avenues for locating nodes in the address space.

5.2.6.2 Context-aware search

Context aware searching refers to situations where an actor has access to resources, such as public DNS, for reconnaissance purposes when performing host enumeration. Unlike blind searching, context-aware searches leverage any available information in order to improve the probability of discovering the reconnaissance subject's networked devices.

It has been shown from this research that combining passive and active enumeration strategies can produce good results from off-link IPv6 host enumeration exercises. The results of the Adaptive-1 (experiments `Adaptive-1a` and `Adaptive-1b`) imply that search methods that employ passive and active enumeration methods, in conjunction with machine learning form a robust strategy for host enumeration.

In a real-world situation it would be recommended to incorporate passive and active methods into a host enumeration exercise in order to increase the likelihood of discovering hosts.

5.2.7 Parallel processing

The study has also displayed the benefits of parallel processing systems in conducting time-consuming research. Parallel processing was used successfully to conduct trials involving applying different algorithms to the test datasets. By executing the simulations for an experiment in parallel, the total cost of the time to complete the simulations was reduced considerably.

The total processing time for the recorded search operations carried out during the study was 387,783,500 seconds. This means that approximately 12.3 years of processing time was consumed throughout this research. Without parallel processing capabilities the research could not have been as extensive, since the time costs would have been far too great to overcome.

5.3 Critical review of the research process

This research has revealed a number of significant findings relating to IPv6 and host enumeration. In particular off-link host enumeration search algorithms have been devised and tested, which will pave the way for future research into host enumeration. There are aspects of the research that would have been done differently if the opportunity was presented. In particular rather than focus on the search algorithms, I would have tested a variety of complete host enumeration strategies against live systems. Although the decision was made to focus on the search algorithms it would have been rewarding to contribute a host enumeration toolkit to the research community. In any case converting the search algorithms that were developed for this research into live scanning tools and testing them in a natural environment is a logical progression to this work.

Additionally, the decision was made to utilise the Wilcoxon ranked tests to test the research hypotheses. This decision may have affected the research outcomes, since the Wilcoxon Signed-Ranked test is 95% as efficient as the independent samples t-test (Sheskin, 2000). If the data were normally distributed the one-sample t-test would have been chosen over the Wilcoxon Signed-Ranked test. Likewise the independent sample t-test would have been chosen over the Wilcoxon Ranked-Sum test. Overall, the decision to use the Wilcoxon tests was made because the data were non-parametric and the tests were deemed appropriate.

Another decision that was made that may have affected the research process was the decision to use Python to create the experimental computer programs. This decision was made early on in the research process, since it was simple to realise the algorithms designed for the experiments into Python code. Throughout the pilot studies the Python programs performed acceptably. It wasn't until testing with the actual experimental parameters (such as 2^{32} probes to transmit) that the extent of the time costs became apparent. Even considering the programs were executed on parallel systems some of the experiments took weeks to complete. The process completion times were also potentially impacted by the load on the clustered computer system. Large differences between the maximum and minimum processing times of simulations were

evident in the results. It is likely that these differences arose from variations in the number of tasks being concurrently performed on the clustered computer system. In the future I would more carefully manage the performance of the system conducting the experiments, to ensure that there is no impact on the results.

If I had to perform the research again, I would have chosen to prototype the design of the algorithms using Python, and then use a lower level language such as C or Julia to write all of the programs for conducting the actual research. I would be more mindful of the limitations of a programming language or other materials, when conducting research in the future.

Chapter 6

Conclusion

6.1 Research overview

This thesis sought to identify appropriate algorithms to use in IPv6 host enumeration search strategies. The research also included testing and validating existing techniques for host enumeration for their applicability to IPv6 host enumeration. The testing included techniques that are used against the IPv4 and IPv6 protocols. In particular, this research was concerned with identifying effective search methods for enumerating an off-link 64 bit subnetwork. Appropriate strategies for performing host enumeration have been established for application to IPv4 networks (both on and off-link), as well as on-link enumeration for IPv6 networks. This research topic was identified in Chapter 1, where it was explained that while there are appropriate and accepted methods for on-link host enumeration against IPv6 networks, the same cannot be said for off-link enumeration.

6.1.1 Host enumeration problem space

The literature surrounding the topic has suggested that the address space is too vast to conventionally search using exhaustive methods. The results of this research support, and provide evidence for this claim. It is, with current computational resources, it is too expensive to conduct an exhaustive enumeration of an IPv6 network. Due to these assertions, little attention has been paid to off-link IPv6 host enumeration in the literature.

Whilst three major categories of search techniques have been presented for off-link IPv6 host enumeration, no reported testing has been conducted to determine the efficacy of said techniques. These three categories of search algorithm are; linear (sequential) search, randomisation functions, and pattern-heuristic searching. The linear search technique is commonly used with IPv4 host enumeration tools, and has also been applied to IPv6 host enumeration tools. Randomisation techniques have also been used in IPv4. However, tools that support randomised search in IPv6 are not available. Finally, pattern-heuristics rely on techniques that exploit predictable elements of the IPv6 addressing schemes (such as low range incremental assignments, or hexadecimal word injection) to target predictable addresses. Even though these methods have been inferred from available tools designed to perform the host enumeration task, little information is available that has undergone peer review.

6.1.2 Method and procedure

The study was conducted in five phases. First, a passive host enumeration survey was conducted into the usage habits of IPv6 in real-world scenarios. A DNS enumeration exercise was chosen to conduct the survey. The results were then used to determine the types of algorithms that could be developed and applied to the search problem. The survey phase also provided the IPv6 addresses that formed the target network devices for search experiments.

Second, the subject host discovery algorithms were generated. A total of six unique algorithms were chosen as the subjects of the controlled, laboratory-based experiments. These algorithms were the; linear search; Stripe search; Monte Carlo search; Pattern-based heuristic search; Adaptive heuristic search; and GA search algorithms. These algorithms were designed to perform searches of the target address space in varied ways.

Third, the experiments were developed. The experimental process and accompanying computer programs were constructed in this phase. The experiments consisted of testing a search algorithm against two IPv6 address datasets. Because of this, each experiment was split into two sub-experiments (an `a` and `b` sub-experiment) that tested the subject algorithm against the randomly generated and surveyed IPv6 ad-

dress datasets respectively. The research assumed testing against two target 64 bit IPv6 subnetworks; one where nodes were configured with addresses that were allocated at random, and one where nodes were configured with addresses that were allocated to resemble real world addressing schemes. As such, each experiment was performed against two datasets of valid addresses, one generated stochastically and one generated from the survey phase of the study.

Fourth, the experiments were performed. Each experiment was run on a cluster of computing devices that shared processing resources. The computer programs written in phase 3 were executed in parallel on the computer cluster. The parameters used to control each sub-experiment are detailed in Chapter 3.

Finally the results were gathered from the experiments and analysed. The information gained from the analysis served to provide the answers to the research questions, as well as suggest further recommendations in light of the study.

6.2 Major conclusions and implications of research

6.2.1 Can networking devices be enumerated on 64 bit IPv6 subnetworks using techniques?

The consensus across the literature is that off-link enumeration is not feasible to conduct against IPv6 networks. According to Hinden and Deering (2006), a standard IPv6 subnetwork should be 64 bits long, leaving 64 bits for possible host addresses. At a rate of 1,000,000 generic probes per second, an exhaustive enumeration of a 64 bit subnetwork would take 584,942.42 years. It has been suggested that IPv6 networks are not susceptible to off-link host enumeration. In contrast on-link enumeration can be achieved with little difficulty in IPv6 due to link-local discovery techniques involving multicast groups. Whilst the research supports that an exhaustive enumeration is infeasible, it has been demonstrated that devices on IPv6 networks can be enumerated by off-link actors.

It was observed that if hosts within an IPv6 subnetwork have been configured using pattern-based, or wordy IPv6 addresses, some degree of host enumeration can

be successfully employed. Likewise, if the address space uses low range incremental interface identifiers, there is little difference between performing host discovery against an IPv6 network and an IPv4 network. Off-link IPv6 host enumeration is possible and viable. However, this research has shown that it is not as comprehensive as it can be against IPv4 networks. It is expected that deterministic methods of host configuration will become increasingly less common as the security and privacy issues associated with such methods are exposed.

6.2.2 Are stochastic searching methods more efficient than deterministic searching methods when enumerating IPv6 hosts within a single 64 bit subnetworks?

IPv4 host enumeration strategies often employ randomisation functions to achieve an even distribution of probes across the entire address space. Randomisation and permutation functions attempt to prevent overloading distant networks that are spatially close together. The effectiveness and applicability for the usage of target randomisation functions in IPv6 host enumeration strategies was tested and validated. It was observed that overall the randomisation functions failed to produce any meaningful results, and therefore the stochastic methods of searching were not appropriate for the problem. The deterministic methods performed better overall, and are therefore the recommended approach to developing host enumeration search algorithms for IPv6 searching.

6.2.3 Do stochastic address allocation schemes within a single 64 bit subnetworks inhibit IPv6 host enumeration strategies?

IPv6 introduces a number of address allocation schemes designed to create high entropy host addresses for IPv6 devices. These schemes, including privacy extensions for SLAAC, hash-based addresses and CGAs, aim to protect hosts from host enumeration attempts, as well as prevent the personally identifiable information from being leaked through IPv6 addresses. This security through obscurity measure is claimed to improve the privacy of users and devices since they are less susceptible to active off-link enu-

meration. This research validated these claims by testing the off-link host enumeration search algorithms against two networks, one which was configured using high entropy IIDs, and one that used IIDs gathered in a passive enumeration exercise, representative of the common usage of the protocol. The results of the experiments indicate that the high entropy addresses provide significant protection from active off-link host enumeration.

It is the recommendation of this thesis that operating systems adopt high entropy addressing as the primary means for allocating IPv6 IIDs, rather than using modified EUI-64 IIDs, or sequential addressing. Having IIDs distributed randomly throughout the address space complicates host enumeration to the point of infeasibility, and the protection it offers is not insignificant. However, for network administrators there is a requirement to ensure that adequate inventory is maintained. This means that systems like DNS, IP address management systems and, if required, DHCPv6 servers must be sufficiently utilised. This will assist in easing the administrative burden of locating devices with semi-permanent or even permanent high entropy addresses on an IPv6 network.

6.2.4 Can machine learning search methods be used to enumerate devices on a 64 bit IPv6 subnetwork?

Machine learning techniques have, thus far, not been applied to the problem of host enumeration. This research sought to determine whether machine learning techniques could be successfully applied to the problem. The research tested a GA as well as a decision system that utilised artificial neural networks, in an effort to answer the research question.

It was discovered that machine learning systems can be effective at enumerating hosts on IPv6 networks, and that they are also effective at classifying IIDs based upon how they were constructed. The adaptive heuristic algorithm performed better than almost all of the other algorithms tested, and highlights that machine learning systems can be used successfully in host enumeration search algorithms. The GA did not perform so admirably, and it was concluded that the approach is inappropriate for

search operations without significant changes.

6.2.5 Are machine learning searching methods more efficient than non-machine learning based methods when enumerating IPv6 hosts within a single 64 bit subnetworks?

Although machine learning-based search algorithms were able to enumerate devices on off-link IPv6 subnetworks, the algorithms that did not employ machine learning performed better overall. It was concluded that non-machine learning search algorithms are a more appropriate choice for most IPv6 host enumeration scenarios. Despite this finding the adaptive heuristic search algorithm did perform remarkably well.

Further research is required to improve the performance of machine learning search methods for off-link IPv6 host enumeration. Particular focus on the GA may aid in improving the search algorithm's ability to locate nodes in known-unknown situations.

6.3 Recommendations and future research

From the findings, it is recommended that network administrators wishing to protect their networked devices from unsolicited off-link IPv6 host enumeration attempts should make use of high entropy (randomly generated or CGA-based) IPv6 addresses. These addresses offer the most security against host enumeration attempts, as evidenced by the results of this research.

Additionally, the usage of IPv4-in-IPv6 style addresses, incremental IIDs, wordy addresses, or other predictable address construction methods greatly increases susceptibility to off-link IPv6 host enumeration efforts. For administrators who wish to ensure their devices can be enumerated through host enumeration methods it is recommended that predictable address construction means are used.

Although it was not addressed specifically in this research, it is important to note that one can maintain a high degree of privacy and mitigate host enumeration techniques with a combination of high entropy IID assignments, and a sufficient IP address management system, in addition to proper DNS management.

Future research could see the algorithms tested in this research employed against

live networked devices in a field experiment. Section 3.2 explained why lab experiments were chosen to over field or natural experiments for this study. Now that the algorithms have been developed and tested in a purely controlled environment, the natural progression is to test their effectiveness in an uncontrolled, live environment. In particular, the adaptive, pattern-based heuristic, stripe search and linear search algorithms have shown great potential as candidates for real-world testing.

These algorithms could be used to aid in similar research efforts such as 'Internet Census 2012' (2013) and Heidemann et al. (2008) as well as provide a means for conducting vulnerability assessments of Internet-based devices in a similar vein to Heninger et al. (2012) and Durumeric et al. (2014).

This research also succeeded in providing a sample of a small scale passive host enumeration of IPv6 addresses using a DNS enumeration. This research can be of use in determining how IPv6 is being utilised in the real world by agents that provide public services. The DNS enumeration is expected to be expanded, and form part of ongoing longitudinal research into IPv6 usage patterns. The machine learning address classification system will play an important role in future studies of this nature.

In addition, the IID classification system developed for this research may aid in defensive detection systems such as IDSs or SIEMs. Such systems could employ the technology to assess the address construction class of the actor that activities originate from. This data could assist in providing insight into how threat actors are utilising IPv6 for malicious purposes, and therefore improve detection and response strategies for such events.

Similarly, another potential avenue for research that has arisen from this study is to consider the host enumeration attempts from the destination's perspective, and determine the search method that is being employed. This research could aid in the early detection and prevention of malicious traffic entering a network. The research data could also be used to determine whether there are other search strategies being employed in the real world that have yet to be published.

Finally, passive enumeration strategies are an area of potential future research for IPv6 systems. There is an abundance of information that can be correlated from passive monitoring of network transmissions. When combined with active enumeration,

a comprehensive enumeration strategy may be achievable against IPv6 networks.

6.4 Final thoughts

This thesis has demonstrated that host enumeration is possible in IPv6 networks. It is not safe to assume that deploying IPv6 provides inherent protection against IPv6 host enumeration exercises, especially in critical systems.

The best search algorithms to use are the Linear search, Adaptive search, Pattern search and Stripe search algorithms. These algorithms produced the most positive results, and performed better than the other algorithms that were tested. Further improvements to the fitness function and operations of the GA could see that as another viable alternative. However, it did not perform well during this research.

IPv6 is an emerging technology and one that has the potential to shape the landscape of network communications for many years to come. More research is required to ensure this protocol meets the needs of its users.

References

Alexa Internet, Inc. (2014, October). *Top 1,000,000 sites (updated daily)*. Retrieved October 2, 2014, from http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

APNIC. (2015, January). *IPv4 exhaustion details*. Retrieved January 5, 2015, from http://www.apnic.net/community/ipv4-exhaustion/ipv4-exhaustion-details

Atlasis, A. (2014, September). Chiron - An All-In-One Penetration-Testing Framework for IPv6. In *Brucon*. Ghent. Retrieved from http://u.jimdo.com/www64/o/sfc6326dbff511243/download/m5145a98b92fd412c/1417096039/AAtlasis_Brucon5x5_2014.pdf

Aura, T. (2005, March). Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard).

Australia.gov.au. (n.d.). *A to z list of government sites*. Retrieved October 28, 2014, from http://www.australia.gov.au/directories/australian-government-directories/a-to-z-list-of-government-sites

Bagnulo, M. (2009, June). Hash-Based Addresses (HBA). RFC 5535 (Proposed Standard).

Bao, C., Huitema, C., Bagnulo, M., Boucadair, M. & Li, X. (2010, October). IPv6 Addressing of IPv4/IPv6 Translators. RFC 6052 (Proposed Standard).

Beale, J., Deraison, R., Meer, H., Temmingh, R. & Walt, C. V. D. (2008, May). *Nessus network auditing* (2nd). Syngress Publishing.

Black, J. & Rogaway, P. (2002). Ciphers with arbitrary finite domains. In B. Preneel (Ed.), *Topics in cryptology — ct-rsa 2002* (Vol. 2271, pp. 114–130). Lecture Notes in Computer Science. Springer Berlin Heidelberg. doi:10.1007/3-540-45760-7_9

Braden, R. (1989, October). Requirements for Internet Hosts - Application and Support. RFC 1123 (INTERNET STANDARD).

Braunton, G. (2005, September). *B.A.S.E. - A Security Assessment Methodology* (Whitepaper No. GSEC Practical Assignment Version 1.4b, Option 1). SANS Institute. Retrieved December 8, 2014, from http://www.sans.org/reading-room/whitepapers/auditing/base-security-assessment-methodology-1587

Brownlee, J. (2011, January). *Clever algorithms: nature-inspired programming recipes* (1st). LuLu.

Carpene, C., Johnstone, M. & Woodward, A. (IN PRESS, December). The effectiveness of classification algorithms on IPv6 IID construction. *International Journal of Autonomous and Adaptive Communications Systems*, *Vol. 7*(4).

Carpene, C. & Woodward, A. (2012, December). Exposing potential privacy issues with IPv6 address construction. In *10th australian information security management conference*. Perth: Edith Cowan University Security Research Institute.

Carpene, C. & Woodward, A. (2014, December). A survey of IPv6 address usage in the public domain name system. In M. Johnstone (Ed.), *The Proceedings of 12th Australian Information Security Management Conference* (ISBN 978-0-7298-0718-0, pp. 91–99). School of Computer & Security Science. Joondalup, WA: Edith Cowan University Security Research Institute.

Cerf, V., Dalal, Y. & Sunshine, C. (1974, December). Specification of Internet Transmission Control Program. RFC 675.

Chown, T. (2008, March). IPv6 Implications for Network Scanning. RFC 5157 (Informational).

Corder, G. W. & Foreman, D. I. (2011). *Nonparametric statistics for non-statisticians: a step-by-step approach*. Wiley. doi:10.1111/j.1751-5823.2010.00122_6.x

Crawford, M. & Haberman, B. (2006, August). IPv6 Node Information Queries. RFC 4620 (Experimental).

Creswell, J. W. (2009). *Research design: qualitative, quantitative, and mixed methods approaches* (3rd). California: Sage Publications, Inc.

de Velde, G. V., Popoviciu, C., Chown, T., Bonness, O. & Hahn, C. (2008, December). IPv6 Unicast Address Assignment Considerations. RFC 5375 (Informational).

Deering, S. & Hinden, R. (1998, December). Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard).

Droms, R. (2004, April). Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6. RFC 3736 (Proposed Standard).

Dunlop, M., Groat, S., Marchany, R. & Tront, J. (2011, February). The good, the bad, the ipv6. In *Communication networks and services research conference (cnsr), 2011 ninth annual* (pp. 77–84). IEEE. doi:10.1109/CNSR.2011.20

Dunlop, M., Groat, S., Urbanski, W., Marchany, R. & Tront, J. (2011, November). MT6D: A Moving Target IPv6 Defense. In *Military communications conference, 2011 - milcom 2011* (pp. 1321–1326). Baltimore, MD: IEEE. doi:10.1109/MILCOM.2011.6127486

Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., ... Payer, M. et al. (2014). The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference* (pp. 475–488). ACM. Retrieved February 17, 2015, from https://jhalderm.com/pub/papers/heartbleed-imc14.pdf

Durumeric, Z., Wustrow, E. & Halderman, J. A. (2013a, August). ZMap: Fast Internet-wide scanning and its security applications. In *Proceedings of the 22nd usenix security symposium (usenix security 13)*. Washington, D.C.: USENIX. Retrieved from https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric

Durumeric, Z., Wustrow, E. & Halderman, J. A. (2013b, August). *Zmap internet scanner*. Retrieved May 13, 2014, from https://github.com/zmap/zmap

Fien, J. (2002, June). Advancing sustainability in higher education. *Advancing sustainability in higher education: issues and opportunities for research*, *3*(3), 243–253. doi:10.1108/14676370210434705

Glassman, M. & Kang, M. J. (2012). Intelligence in the internet age: the emergence and evolution of open source intelligence (osint). *Computers in Human Behavior*, *28*(2), 673–682. doi:10.1016/j.chb.2011.11.014

Gont, F. (2014, April). A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC). RFC 7217 (Proposed Standard).

Graham, R. (2012, October). *Randomizing port scans, part two*. Errata Security. Retrieved July 4, 2013, from http://blog.erratasec.com/2012/10/randomizing-port-scans-part-two.html

Graham, R. (2013a, December). *Masscan: designing my own crypto*. Retrieved May 13, 2014, from http://blog.erratasec.com/2013/12/masscan-designing-my-own-crypto.html

Graham, R. (2013b, September). *Masscan: mass IP port scanner*. Retrieved August 29, 2014, from https://github.com/robertdavidgraham/masscan

Graham, R. (2013c, September). *Masscan: the entire Internet in 3 minutes*. Errata Security Blog. Retrieved May 13, 2014, from http://blog.erratasec.com/2013/09/masscan-entire-internet-in-3-minutes.html

Graziano, A. M. & Raulin, M. L. (2004). *Research methods: a process of inquiry*. (5th). Pearson Education Group.

Groat, S., Dunlop, M., Marchany, R. & Tront, J. (2010). The privacy implications of stateless IPv6 addressing. In *Proceedings of the sixth annual workshop on cyber security and information intelligence research* (52:1–52:4). CSIIRW '10. Oak Ridge, Tennessee, USA: ACM. doi:10.1145/1852666.1852723

Groat, S., Dunlop, M., Marchany, R. & Tront, J. (2011). IPv6: nowhere to run, nowhere to hide. In *Proceedings of the 2011 44th hawaii international conference on system sciences* (pp. 1–10). HICSS '11. Washington, DC, USA: IEEE Computer Society. doi:10.1109/HICSS.2011.258

Habets, T. (2009). *Arping v2.15*. Retrieved January 23, 2015, from http://github.com/ThomasHabets/arping

Hastie, T., Tibshirani, R. & Friedman, J. (2013, January). *Data mining, inference, and prediction* (2nd). The elements of statistical learning. Springer.

Hauser, V. (2006). Attacking the IPv6 Protocol Suite. In *Pacsec 2005*. The Hacker's Choice. Retrieved from https://www.thc.org/papers/vh_thc-ipv6_attack.pdf

Hauser, V. (2014, December). *THC-IPv6*. Retrieved December 31, 2014, from https://www.thc.org/thc-ipv6/

Heidemann, J., Pradkin, Y., Govindan, R., Papadopoulos, C., Bartlett, G. & Bannister, J. (2008). Census and survey of the visible internet. In *Proceedings of the 8th acm sigcomm conference on internet measurement* (pp. 169–182). ACM.

Heninger, N., Durumeric, Z., Wustrow, E. & Halderman, J. A. (2012). Mining your Ps and Qs: detection of widespread weak keys in network devices. In *Usenix security symposium* (pp. 205–220). Bellevue, WA: USENIX. Retrieved from https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger

Hinden, R. & Deering, S. (2006). IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard).

Hinden, R., Deering, S. & Nordmark, E. (2003, August). IPv6 Global Unicast Address Format. RFC 3587 (Informational).

Hold-Geoffroy, Y., Gagnon, O. & Parizeau, M. (2014). Once you SCOOP, no need to fork. In *Proceedings of the 2014 annual conference on extreme science and engineering discovery environment* (p. 60). ACM. Retrieved from https://code.google.com/p/scoop/

Hunter, J. D. (2007, June). Matplotlib: a 2D graphics environment. *Computing In Science and Engineering*, *9*(3), 90–95. doi:10.1109/MCSE.2007.55

IANA. (2013, February). *Internet protocol version 6 address space*. Retrieved February 17, 2015, from http://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml

IANA. (2014a, October). *Internet Protocol Version 4 Address Space Registry*. Retrieved February 17, 2015, from http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml

IANA. (2014b, May). *IPv6 Global Unicast Address Assignments*. Retrieved February 17, 2015, from http://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml

IANA. (2015, February). *IPv6 Multicast Address Space Registry*. Retrieved February 17, 2015, from http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml

IEEE Standards Association. (n.d.). *Guidelines for 64-bit Global Identifier (EUI-64).* IEEE.

Jackson, S. (2009). *Research methods and statistics: a critical thinking approach* (3rd). Belmont, CA: Cengage Learning.

Jara, A. J., Ladid, L. & Skarmeta, A. (2013, September). The internet of everything through IPv6: an analysis of challenges, solutions and opportunities. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, *4*(3), 97–118.

Kawamura, S. & Kawashima, M. (2010, August). A Recommendation for IPv6 Address Text Representation. RFC 5952 (Proposed Standard).

Kessler, G. (2014, November). *An overview of TCP/IP protocols and the internet.* Gary Kessler Associates.

Kim, D. & Solomon, M. (2010, November). *Fundamentals of information systems security.* Burlington, MA: Jones & Bartlett Learning.

Klensin, J. (2001, April). Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard).

Klensin, J. (2004, February). Application Techniques for Checking and Transformation of Names. RFC 3696 (Informational).

Knuth, D. E. (1998). *Art of computer programming, volume 3: sorting and searching, the* (2nd). Addison-Wesley.

Kumar, A., Paxson, V. & Weaver, N. (2005). Exploiting underlying structure for detailed reconstruction of an Internet-scale event. In *Proceedings of the 5th acm sigcomm conference on internet measurement* (pp. 33–33). IMC '05. Berkeley, CA, USA: USENIX Association.

Lacey, S. & Box, R. (n.d.). *A fast easy sort.* Retrieved January 6, 2015, from http://cs.clackamas.cc.or.us/molatore/cs260Spr03/combsort.htm

Leonard, D. & Loguinov, D. (2010). Demystifying service discovery: implementing an internet-wide scanner. In *Proceedings of the 10th acm sigcomm conference on internet measurement* (pp. 109–122). ACM. doi:10.1145/1879141.1879156

Lyon, G. (2015, January). Nmap SVN source code repository. Computer Program. Retrieved from https://svn.nmap.org/

Lyon, G. F. (2009). *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. USA: Insecure.

Mackenzie, N. & Knipe, S. (2006). Research dilemmas: paradigms, methods and methodology. *Issues in educational research*, *16*(2), 193–205.

McKinney, W. (2010, July). Data structures for statistical computing in python. In *Proceedings of the 9th python in science conference (scipy 2010)* (pp. 51–56). Austin, Texas.

McKinney, W. (2012, November). *Python for data analysis: data wrangling with pandas, numpy, and ipython*. Sebastopol, CA: O'Reilly Media, Inc.

McNab, C. (2007, November). *Network security assessment: know your network*. O'Reilly Media.

Mertens, D. M. [Donna M.]. (2007, July). Transformative paradigm: mixed methods and social justice. *Journal of Mixed Methods Research*, *1*(3), 212–225. doi:10.1177/1558689807302811

Mertens, D. M. [Donna M]. (2010, April). Philosophy in mixed methods teaching: the transformative paradigm as illustration. *International Journal of Multiple Research Approaches*, *4*(1), 9–18. doi:10.5172/mra.2010.4.1.009

Montgomery, D. C. (2009). *Design and analysis of experiments* (7th). Hoboken, NJ: Wiley.

Narten, T., Draves, R. & Krishnan, S. (2007, September). Privacy Extensions for Stateless Address Autoconfiguration in IPv6. RFC 4941 (Draft Standard).

Net-SNMP. (2011, May). *IPv6MIB*. Retrieved January 20, 2015, from http://www.net-snmp.org/docs/mibs/ipv6MIB.html

Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. (1994). ISO. Retrieved from http://www.iso.org/iso/catalogue_detail.htm?csnumber=20269

Internet Census 2012. (2013). Retrieved February 17, 2015, from http://internetcensus2012.bitbucket.org/paper.html

OTB Development Team. (2014, September). *The orfeo tool box software guide*. CNES.

Pérez, F. & Granger, B. E. (2007, May). IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, *9*(3), 21–29. doi:10.1109/MCSE.2007.53

Polcák, L. (2014). Challenges in identification in future computer networks. In *Icete 2014 doctoral consortium* (pp. 15–24). Wien, AT: SciTePress - Science and Technology Publications. Retrieved from http://www.fit.vutbr.cz/research/view_pub.php?id=10516

Porras, P., Saidi, H. & Yegneswaran, V. (2009, March). *An analysis of conficker's logic and rendezvous points*. SRI International. Menlo Park, CA. Retrieved February 20, 2015, from http://mtc.sri.com/Conficker/

Postel, J. (1981, September). Internet Protocol. RFC 791 (INTERNET STANDARD).

Potter, S. (2006). *Doing postgraduate research* (2nd). London: Sage.

Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J. & Lear, E. (1996, February). Address Allocation for Private Internets. RFC 1918 (Best Current Practice).

Rish, I. (2001). An empirical study of the naive bayes classifier. In *Ijcai 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, *22*, pp. 41–46). doi:10.1.1.330.2788

Rivest, R. L. & Schuldt, J. C. N. (2014, October). Spritz—a spongy RC4-like stream cipher and hash function. Presented at CRYPTO 2014 Rump Session. Retrieved February 20, 2015, from http://people.csail.mit.edu/rivest/pubs/RS14.pdf

RStudio. (2014). RStudio: integrated development environment for R (version 0.98.1102). Computer Program. Boston, MA. Retrieved January 21, 2015, from http://www.rstudio.org/

Savagaonkar, U., Sahita, R., Nagabhushan, G., Rajagopal, P. & Durham, D. (2005). An os independent heuristics-based worm-containment system. Retrieved February 20, 2015, from http://vxheaven.org/lib/pdf/An%20OS%20Independent%20Heuristics-based%20Worm-containment%20System.pdf

Schemers, R. (2012, October). *Fping*. Retrieved February 20, 2015, from http://fping.org/

Schemers, R. & Schweikert, D. (2014, May). *Fping*. Retrieved February 20, 2015, from http://fping.org/

Schneider, K.-M. (2003). A comparison of event models for naive bayes anti-spam e-mail filtering. In *Proceedings of the tenth conference on european chapter of the association for computational linguistics - volume 1* (pp. 307–314). EACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics. doi:10.3115/1067807.1067848

scikit-learn developers. (n.d.). *Machine learning 101: general concepts*. Retrieved January 31, 2015, from http://www.astroml.org/sklearn_tutorial/general_concepts.html

Shannon, C. & Moore, D. (2004, July). The spread of the witty worm. *Security Privacy, IEEE*, *2*(4), 46–50. doi:10.1109/MSP.2004.59

Sheskin, D. J. (2000). *Parametric and nonparametric statistical procedures* (Second). Florida: Chapman & Hall/CRC.

Sullo, C. & Lodge, D. (2015). *Nikto*. CIRT.net. Retrieved February 17, 2015, from https://github.com/sullo/nikto

Thomson, S. & Narten, T. (1998, December). IPv6 Stateless Address Autoconfiguration. RFC 2462 (Draft Standard).

Thomson, S., Narten, T. & Jinmei, T. (2007, September). IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard).

Wagner, J.-O., Wiegand, M., Brown, T. & Mauthe, C. K. (2009, January). *OpenVAS Compendium* (1st). Germany: Intevation GmbH.

Walt, S. v. d., Colbert, S. C. & Varoquaux, G. (2011, March). The numpy array: a structure for efficient numerical computation. *Computing in Science and Engineering*, *13*(2), 22–30. doi:10.1109/MCSE.2011.37

Wei-hua, J., Wei-hua, L. & Jun, D. (2003, August). The application of ICMP protocol in network scanning. In *Parallel and distributed computing, applications and technologies, 2003. pdcat 2003. proceedings of the fourth international conference on* (pp. 904–906). IEEE. doi:10.1109/PDCAT.2003.1236446

Weisstein, E. W. (n.d.). *Primitive root*. MathWorld–A Wolfram Web Resource. Retrieved February 20, 2015, from http://mathworld.wolfram.com/PrimitiveRoot.html

Williamson, K. & Johanson, G. (2013). *Research methods: information, systems and contexts* (1st). Prahran, VIC: Tilde University Press.

xslidian & VersusClyne. (2014, October). *IPv6 hosts*. Retrieved February 20, 2015, from https://code.google.com/p/ipv6-hosts/

Yu, Z. & Tsai, J. (2011, January). *Intrusion detection: a machine learning approach* (3rd). Series in electrical and computer engineering. Imperial College Press.

Zalewski, M. (2005). *Silence on the wire: a field guide to passive reconnaissance and indirect attacks*. San Francisco, CA, USA: No Starch Press.

Zanella, A., Bui, N., Castellani, A., Vangelista, L. & Zorzi, M. (2014, February). Internet of things for smart cities. *Internet of Things Journal, IEEE*, *1*(1), 22–32. doi:10.1109/JIOT.2014.2306328

Glossary

ANN Artificial neural network. A type of machine learning system that simulates the human brain. 49, 50, 51, 85, 110

ARP Address Resolution Protocol. A network-layer discovery protocol used within IPv4 to resolve link-layer addresses. 4, 13, 26, 36, 37, 38, 39, 44, 45

BYOD Bring your own device. A policy relating to the use of arbitrary unmanaged personal devices on corporate networks. 3

CGA Cryptographically Generated Address. A method for generating IPv6 addresses with cryptographic properties that is defined in RFC-3972 (Aura, 2005). CGAs use a portion of the host's public key to enable the addresses to be authenticated by other devices. 23, 25, 72, 200, 212, 226

DNS Domain name system. A system used to map IP numbers to host names. 37, 42, 43, 44, 45, 46, 70, 71, 109, 224, 228, 229

GA Genetic algorithm. An evolutionary computing algorithm that simulates biological evolutionary processes. xvii, 77, 91, 101, 102, 158, 159, 163, 167, 169, 213, 224, 227, 228, 230

HBA Hash-Based Address. A method for generating IPv6 addresses with cryptographic properties that is defined in RFC-5535 (Bagnulo, 2009). HBAs apply a cryptographic hashing function to generate a visually random address. 25, 200, 212

host enumeration The act of locating active nodes on a computer network. 1, 52, 223, 224, 225, 226, 227, 228, 229, 230

ICMPv4 Internet Control Message Protocol version 4. A messaging protocol used within IPv4 for communications at TCP/IP layer 2 and OSI layer 3. 40

ICMPv6 Internet Control Message Protocol version 6. A messaging protocol used heavily within IPv6 for communications at TCP/IP layer 2 and OSI layer 3. 4, 40, 41, 58

IID Interface Identifier. The host portion of an IPv6 address. According to RFC-4291 (Hinden & Deering, 2006), the IID should be 64 bits long. xi, xii, xv, xviii, 17, 20, 22, 23, 24, 25, 36, 49, 50, 51, 57, 80, 85, 89, 109, 110, 113, 130, 132, 142, 156, 157, 162, 173, 177, 184, 185, 200, 211, 212, 216, 217, 218, 226, 227, 228, 229

IoT Internet of Things. An emerging technology that involves autonomous machine-to-machine communications. 3

IPv4 Internet Protocol version 4. An OSI layer 3, TCP/IP Layer 2 networking protocol. IPv4 uses 32 bits to address network nodes. i, 1, 2, 3, 4, 5, 13, 14, 15, 16, 18, 20, 21, 26, 28, 29, 31, 33, 34, 38, 40, 44, 45, 46, 47, 52, 54, 55, 56, 57, 223, 225, 226

IPv6 Internet Protocol version 6. An OSI layer 3, TCP/IP Layer 2 networking protocol. IPv6 uses 128 bits to address network nodes. i, ii, vii, viii, xi, xvii, 1, 2, 4, 5, 6, 7, 8, 9, 13, 14, 15, 16, 17, 18, 17, 18, 19, 20, 21, 22, 23, 26, 27, 28, 35, 36, 38, 39, 41, 42, 44, 45, 46, 47, 49, 50, 51, 52, 54, 55, 56, 57, 58, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 77, 80, 83, 85, 86, 89, 90, 94, 97, 98, 101, 102, 103, 109, 110, 113, 115, 116, 126, 133, 137, 145, 152, 158, 159, 163, 173, 176, 180, 195, 200, 201, 202, 203, 204, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 223, 224, 225, 226, 227, 228, 229, 230

NAT Network Address Translation. A mechanism to map many network addresses to one or many network addresses. Prominently used with IPv4 to prevent exhaustion of the address space resources. 1, 58

NDP Neighbour Discovery Protocol. IPv6's discovery protocol used to enable communications between IPv6 enabled devices. 36, 39

NIQ Node Information Query. An ICMP message type that requests information from another IP enabled device.. 37, 41

SLAAC Stateless Address Auto-configuration. A method defined in RFC-2462 (Thomson & Narten, 1998) for devices to generate their own unique IPv6 addresses statelessly. 22, 23, 24, 57, 58, 200, 211, 216, 217

subnetwork An IP network that has been divided. The term is used interchangeably with subnet. 1, 14, 17, 20, 26, 38, 40, 47, 56, 223, 225

VA Vulnerability assessment. A VA is used to assess a system for vulnerabilities to threats. i, 2, 3, 41, 52, 53, 54