BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

COMPARISON OF IPV6 PREFIX SET GENERATORS

SROVNÁNÍ GENERÁTORŮ IPV6 PREFIXOVÝCH SAD

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE
DOMINIK VAŠEK

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. JIŘÍ MATOUŠEK

BRNO 2018

Zadání bakalářské práce/20801/2017/xvasek06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel:	**Vašek Dominik**
Obor:	Informační technologie
Téma:	**Srovnání generátorů IPv6 prefixových sad**
	Comparison of IPv6 Prefix Set Generators
Kategorie:	Počítačové sítě

Pokyny:

1. Seznamte se se síťovým protokolem IPv6 a s důvody pro generování sad prefixů IPv6 adres.
2. Nastudujte existující generátory prefixů IPv6 adres.
3. Navrhněte vhodnou množinu kritérií využitelných k porovnání vlastností existujících generátorů.
4. Implementujte sadu skriptů sloužících k porovnání existujících generátorů podle navržených kritérií.
5. Proveďte porovnání existujících generátorů a diskutujte dosažené výsledky.

Literatura:

- M. Lorenc: Generátor IPv6 tabulek, bakalářská práce, Brno, FIT VUT v Brně, 2013.
- K. Zheng and B. Liu, "V6Gene: A Scalable IPv6 Prefix Generator for Route Lookup Algorithm Benchmark," in Proc. of the 20th International Conference on Advanced Information Networking and Applications. IEEE Computer Society, 2006, pp. 147-152, ISBN 0-7695-2466-4-01.
- M. Wang, S. Deering, T. Hain, and L. Dunn, "Non-random Generator for IPv6 Tables," in Proc. of the 12th Annual IEEE Symposium on High Performance Interconnects. IEEE Computer Society, 2004, pp. 35-40, ISBN 0-7803-8686-8.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Vedoucí:	**Matoušek Jiří, Ing.**, UPSY FIT VUT
Datum zadání:	1. listopadu 2017
Datum odevzdání:	16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

[illegible]

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstract

This bachelor's thesis aims to compare the IPv6 generators. In the first part we introduce protocol IPv6 and the allocation policies of registrars in order to better understand the problem. Next, we introduce three existing prefix set generators, whose description is available. In the second part of the thesis we propose and implement a series of tests for the prefix set generators. Lastly, we test the generators and make a conclusion based on the results. According to our results, we conclude that the prefix sets from the NonRandom and IPv6Table generators have a large error in comparison with a target prefix set. However, V6Gene, the implementation of which is not currently publicly available, might generate prefix sets close to the reality based on its proposal.

Abstrakt

Cílem této bakalářské práce je porovnání generátorů IPv6 prefixových sad. V první části práce je představen protokol IPv6 a politiky přidělování IPv6 registrátorů, které tvoří základ řešené problematiky. Následně jsou zmíněny tři existující generátory prefixových sad, jejichž popis je k dispozici. V druhé části je navržena a implementována série testů pro porovnávání generátorů prefixových sad. V poslední části jsou provedeny testy prefixových sad na základě dříve definovaných kritérií. Na základě provedených testů byl vyvozen závěr, že prefixové sady vygenerované generátory IPv6Table a NonRandom, jejichž implementace je veřejně dostupná, vykazují velkou chybu v porovnání s realitou. S ohledem na popis generátoru V6Gene lze očekávat, že prefixové sady vygenerované tímto generátorem by měly dosahovat výrazně menší chyby v porovnání s realitou. Implementace tohoto generátoru však není veřejně dostupná, takže tato očekávání nebylo možné experimentálně ověřit.

Keywords

IPv6 prefix set generator, comparison, IPv6 address allocation policies

Klíčová slova

generátor IPv6 prefixových sad, porovnání, politiky přidělování IPv6 adres

Reference

VAŠEK, Dominik. *Comparison of IPv6 Prefix Set Generators.* Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Matoušek

Rozšířený abstrakt

S blížícím se vyčerpáním IPv4 adresového prostoru bylo potřeba vymyslet náhradu, kterou se stal protokol IPv6. Ten má v porovnání s protokolem IPv4 mnohem větší adresový prostor. Díky zkušenostem z protokolu IPv4 bylo v protokolu IPv6 možno vylepšit i řadu jiných nedostatků, jako například zabezpečení, průhlednost a především velikost adresového prostoru. Kvůli velikosti adresového prostoru se však objevil další problém a tím je potřeba výrazně efektivnějších směrovacích algoritmů v porovnání s protokolem IPv4. Vzhledem k tomu, že dosavadní IPv6 směrovací tabulky nejsou příliš velké, není možné tyto nové směrovací algoritmy plně otestovat na reálných prefixových sadách. Namísto toho se využívají generované prefixové sady, které však ne vždy dostatečně přesně odrážejí skutečnost. Proto se tato bakalářská práce zabývá návrhem kritérií a implementací nástroje pro porovnávání generátorů IPv6 prefixových sad.

Druhá kapitola se zabývá samotným protokolem IPv6. Jak již bylo zmíněno, hlavním rozdílem protokolu IPv6 oproti protokolu IPv4 je jeho adresový prostor, který obsahuje 2^{128} adres oproti 2^{32} adres protokolu IPv4. Tím vznikají nové problémy, jako například délka zapisované IPv6 adresy. Existují mechanismy pro zkrácení tohoto zápisu. Příkladem může být vynechání nejlevějších nul v jednotlivých segmentech nebo záměna série segmentů za dvě dvojtečky. Samotná adresa se pak skládá ze dvou částí. První částí je prefix, který určuje síť, do které adresa spadá, a je přidělován registrátory. Druhá část pak označuje konkrétní zařízení v dané síti. Vzhledem k množství adres a potřebě přidělovat adresy po celém světě byla již u protokolu IPv4 definována hierarchie registrátorů. Jednotliví registrátoři spravují a přidělují různé délky prefixů. Jejich hlavním úkolem je zajistit vyvážené přidělování adresového prostoru zamezující nutnost přepočítávání adresového prostoru v budoucnu a zároveň šetření adresovým prostorem, aby nedošlo k jeho rychlému vyčerpání, jako v případě protokolu IPv4. V dnešní době jsou přidělovány především prefixy délky `/32` a `/48`. V budoucnu však můžeme očekávat především nárůst počtu prefixů délky `/64`, což povede k velké změně v rozložení délek prefixů, na kterou je vhodné se připravit efektivními směrovacími algoritmy.

Ve třetí kapitole jsou popsány tři existující generátory prefixových sad, jejichž popis je volně dostupný. Prvním z nich je generátor V6Gene, který se zaměřuje na generování prefixových sad na principu prefixového stromu. Celkově generování prefixů probíhá ve dvou fázích: přidělování nových prefixů do prefixového stromu následované náhodným rozgenerováním již existujících prefixů. Druhým generátorem je NonRandom generátor, který pro změnu využívá existujících IPv4 prefixových sad, jejichž prefixy jsou zdvojnásobeny v délce, aby reflektovaly rozložení IPv6 délek prefixů. Posledním popisovaným generátorem je IPv6Table generátor, který si z existující IPv6 prefixové sady spočítá pravděpodobnostní rozložení jednotlivých délek a distribuce bitů, na základě kterých začne generovat požadovanou prefixovou sadu.

Čtvrtá kapitola se zaměřuje na návrh kritérií pro porovnávání prefixových sad. Navrženy jsou skupiny testů. První skupinou jsou testy zabývající se základními vlastnostmi prefixových sad, a sice rozložení délek prefixů a rozložení hodnot bitů. Druhou skupinou jsou testy nad prefixovým stromem, konkrétně zanoření prefixů do sebe, větvení prefixového stromu a vyvážení prefixového stromu. Poslední skupinou jsou testy zabývající se efektivitou samotného generátoru. Tato skupina sleduje množství duplicit v generované sadě, ale také celkové hardwarové nároky generátoru.

Pátá kapitola se zabývá implementací nástroje využívajícího kritéria specifikovaná v předešlé kapitole. Je navržen způsob testování založený na využití historických prefixových sad, na základě kterých probíhá generování prefixových sad, jež jsou přibližně stejně velké jako

aktuální prefixové sady. Pro porovnání výsledků je navrženo využívat tzv. RMSE, který porovnává průměrnou chybu mezi generovanými a cílovými prefixovými sadami.

Poslední kapitola se zabývá samotným porovnáním generátorů prefixových sad, jejichž implementace je veřejně dostupná. Na základě testů a návrhu generátorů bylo zjištěno, že ani jeden ze dvou testovaných generátorů nebere v potaz politiky přidělování adres, ale pouze rozgenerovává danou sadu. To znamená, že podle velikosti časového úseku mezi prefixovými sadami se bude chyba lišit. Poslední generátor V6Gene, který nebylo možné otestovat, jelikož jeho implementace není momentálně veřejně dostupná, by na základě návrhu a pozorování ostatních generátorů měl teoreticky z velké míry generovat prefixové sady podobné sadám v budoucnu.

Comparison of IPv6 Prefix Set Generators

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Jiří Matoušek. All the relevant information sources, which were used during the preparation of this thesis, are properly cited and included in the list of references.

....
Dominik Vašek
May 16, 2018

Acknowledgements

I would like to thank my supervisor Ing. Jiří Matoušek for his time, guidance and valuable input on the thesis.

Contents

Chapter 1

Introduction

Due to the upcoming vision of exhaustion of the IPv4 address space, it was decided that it is necessary to find a replacement for this protocol at the beginning of the 1990s. The solution was protocol IPv6 [7], which has much larger address space in comparison with IPv4 (2^{128} compared to 2^{32}). In the upcoming years, there were many new definitions of the protocol [18][7], which made it easier to adopt. However, mostly due to the need of software and hardware support of IPv6, there were other attempts to find a solution. The first solution was humbler allocation policies and the second, most essential, was NAT (Network Address Translation) which accounted to the massive delay of the full IPv6 deployment. [12]

Routing in the vast address space of IPv6 requires significantly more effective routing algorithms than those used in the IPv4 networks. Moreover, the new protocol has also larger hardware requirements. However, validation and testing of hardware and software for this protocol is not completely possible because current IPv6 routing tables do not contain enough addresses to fully test every aspect of routing algorithms in a large scale. This problem can be however fixed by generating a prefix set that is large enough for the testing of hardware devices and routing algorithms.

This bachelor's thesis is dedicated to the comparison of individual prefix set generators to real-life samples. The goal is to compare each generator with real prefix sets and identify their advantages in comparison to other generators.

Chapter 2 presents protocol IPv6, reasons for its deployment and the hierarchy of address allocation registrars. In Chapter 3 we describe three prefix set generators which description is available. Chapter 4 introduces possible testing criteria divided into three section. The first section proposes basic prefix set properties: prefix length distribution and bit value distribution; the second section proposes trie properties: prefix nesting, branching of trie and skew of trie. Finally, the third section proposes performance comparison of generators. In the first section of Chapter 5 we proposed the means of testing using criteria described in Chapter 4 and described the workflow of a tool for comparing IPv6 prefix set generators. In second section of Chapter 5 we have described the implementation of the tool. In the first section of Chapter 6, different generators are compared to each other and existing prefix sets using the criteria first introduced in Chapter 4. In the second section of Chapter 6 we summarize the results we have found in the first section. Chapter 7 contains the conclusion of this bachelor's thesis.

Chapter 2

Protocol IPv6

Protocol IPv6 is a communication protocol whose objective is to create unique identification within the Internet network. Protocol IPv6 is supposed to completely replace current protocol IPv4, which primarily suffers from small address space. [9]

2.1 Grounds for IPv6

The predecessor of protocol IPv6 is protocol IPv4. The IPv4 protocol has an address space of 2^{32} addresses. However, this address space also contains reserved blocks. At the beginning of address space allocation, its capacity was not much taken into account. This led to quick address space exhaustion in the first decade of IPv4 usage. Due to this reason, it was necessary to redefine an allocation policy. It was finally decided to abandon classful routing, which divided addresses into classes A, B, and C for unicast addresses. These classes differ from one another in the length of an address mask (`8`, `16`, and `24` bits). This led to the introduction of classless routing, which allowed different prefix lengths. Classless routing thus enabled allocation of the amount of addresses actually necessary for different objects, instead of large chunks of the address space. Even though this solution dramatically reduced the demand on the address space, the solution was still insufficient due to the increasing number of new devices connected. This led to the need for a new protocol. The new protocol was introduced in 1998 as protocol IPv6, which allows to address up to 2^{128} devices.

2.2 IPv6 address

In protocol IPv6, an address is represented by 128 bits. The address is noted in hexadecimal and divided into eight segments. Each segment consists of four hexadecimal numbers, which represent two octets. If we consider the length of IPv6 addresses, it is vital for us to shorten the address as much as possible. There are several possible ways of shortening the address. The first one is reducing the amount of leftmost zeros of each segment. For example, address

```
2001:0db8:0000:0100:0000:0000:2378:ca42
```

can be shortened like this

```
2001:db8:0:100:0:0:2378:ca42
```

An address in this format can be shortened even further. One or multiple segments consisting of zeros can be replaced by two colons "`::`", reducing the length of written address. This can be however done only once per address, because it has to be possible to correctly restore the original address for routing. The address can be thus minimized like this

```
2001:db8:0:100::2378:ca42
```

The minimization is usually done between the most octets composed of zeros or, if there are two same blocks of zeros, from left.

2.3 Prefixes

An IPv6 address consists of two parts. The first part determines a network and the second part determines a host inside the network. This division is determined by a network prefix, which is written at the end of the address, after a slash. If we consider the address

```
2001:0db8:1280:1::0001/64
```

we can say that `2001:0db8:1280:1::/64` is a network identifier and the whole address represents a particular host inside the network. Another interesting fact is that this prefix can be a part of another network, thus prefix. This means that from a parent network `2001:0db8:1280::/48` there can be up to $2^{16}-1$ subnetworks coexisting with our network.

Currently, we distinguish several reserved prefixes [18][7]:

- Unicast – `2000::/3` – currently the only prefix reserved for unicast
- Multicast – `ff00::/8`
- Link-local – `fe80::/10`
- ULA – `fc00::/8` – unique local addresses, a prefix is augmented by 40 random bits for the prefix length of `/48`, which minimizes the probability of conflict when joining networks
- Loopback – `::1/128`
- Not specified – `::/128`

2.4 Protocol properties

Protocol IPv6 has many different properties in comparison to protocol IPv4. Some of these properties give the new protocol advantages, which are described below.

2.4.1 Transparency

Due to a massive size of IPv6 address space, it is possible to route directly towards endpoint devices without the need for Network Address Translation (NAT). The protocol also has standardized network flow control via Quality of Service (QoS), as the protocol was already developed with the concept of packet body encryption in mind. [12]

2.4.2 Security

The protocol has a built-in IPSec encryption and integrity check. Along with these, it is also possible to use protocol Secure Neighbor Discovery (SEND), which serves to secure the discovery of neighbor stations, upkeep of the information about accessibility and destination paths. Therefore, mainly thanks to these advantages, protocol IPv6 is not prone to falsifies. [1][12]

2.5 IPv6 address allocation

Currently, IPv6 address allocation consists of two parts. The first is based on prefix allocation, which is controlled by registrars. The second part consists of the allocation of endpoint addresses, which is usually done using auto configuration. To get an endpoint address, it is however necessary to get the prefix of the network first.

2.5.1 Registrar hierarchy

Currently, there are five levels of registrars. By a registrar we mean an organization or institution, which focuses on address space allocation. The root registrar is organization IANA, which releases address space blocks for individual Regional Internet Registries (RIRs). These regional registrars then allocate parts of their address space to National Internet Registries (NIRs) or Local Internet Registries (LIRs). NIRs are mostly common in Asia and can also contain any number of LIRs. LIR and NIRs can further allocate address space to Internet Service Providers (ISPs) or directly to Endpoint Users (EUs). The hierarchy is illustrated in Figure 2.1.

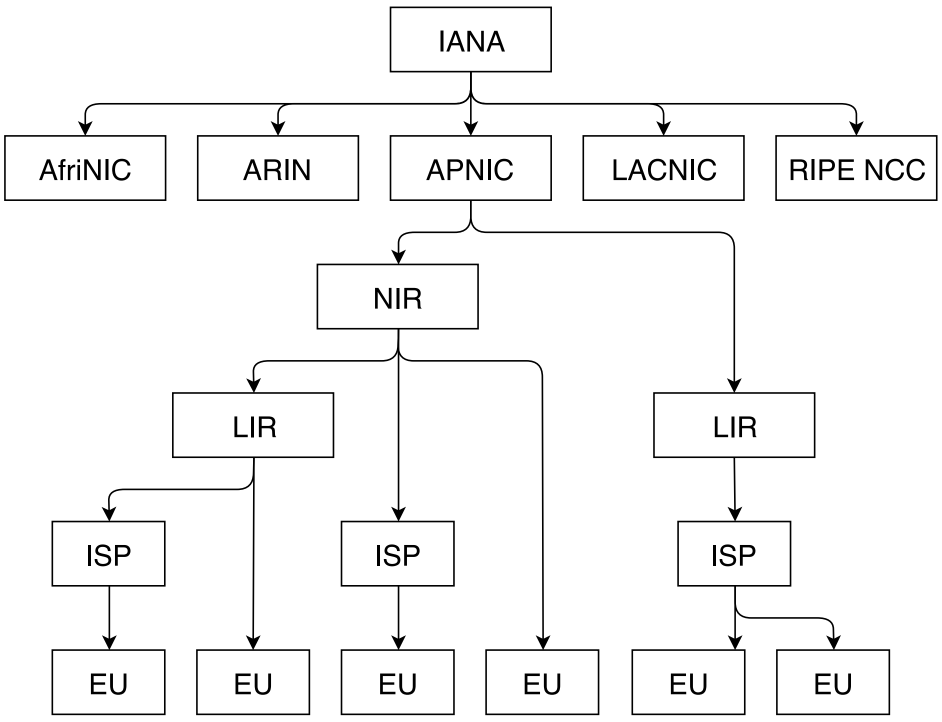

Figure 2.1: Hierarchy of registrars [3].

2.5.2 Address allocation principles

As stated above, there is a whole hierarchy of registrars involved in address space allocation, whose main purpose is to allocate address space in a reasonable manner. This is mainly to limit the speed of address space exhaustion. This subsection contains the description and policies of all registrars.

Internet Assigned Numbers Authority (IANA)
IANA is an organization which coordinates address space allocation. It usually allocates the address space to a specific RIR. The principles of new address space allocation are [11]:

- One address block, which is assigned to a RIR, has the prefix length of `/12`.
- A RIR should get enough addresses that will last at least for 18 months.
- Each new RIR gets automatically one address block.
- A RIR can request a new address block if at least 50 percent of its current address block is allocated.
- A RIR can request a new address block if it does not have enough addresses for the upcoming 9 months.

The calculation of address space N, which is necessary for a RIR to operate without the necessity of further address space allocation is

$$N = A * P$$

where A represents the average number of allocated addresses per month in the past 6 months and P represents the length of a period in months for which the address space of the RIR must endure without further reservation.

Regional Internet Registry (RIR)
RIRs are registrars which represent a large area, such as whole continents (Figure 2.2). Currently, there are five RIRs. Each of them might have different allocation policies.

- AFRINIC – African Network Information Center
- APNIC – Asia-Pacific Network Information Centre
- ARIN – American Registry for Internet Numbers
- LACNIC – Latin America and Caribbean Network Information Centre
- RIPE NCC – Réseaux IP Européens Network Coordination Centre

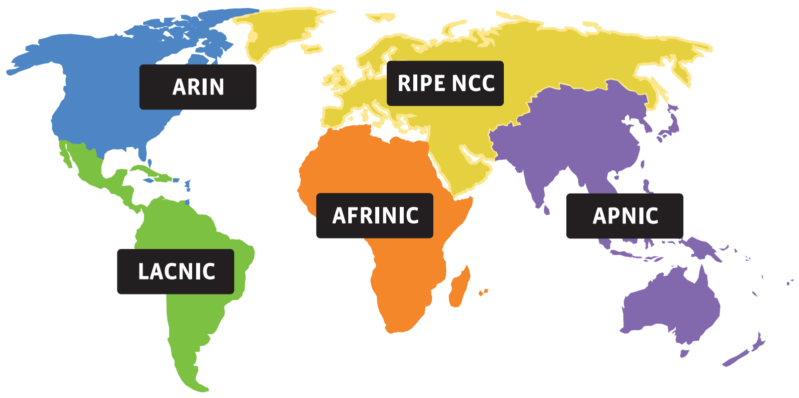

Figure 2.2: Map of Regional Internet Registries [10].

As an example, we can consider the conditions of RIPE NCC [17], which also includes the Czech Republic.

- The smallest allocable block is `/32`.
- To get the first allocation block, it is necessary for an organization to be a LIR or present an allocation plan for the upcoming two years.
- For further address space allocation of `/32` or larger, it is necessary for a registrar to meet Host-Density Ration [6] 0,94 or larger, where

$$HD - Ratio = \frac{log(\text{total allocations})}{log(\text{maximum of possible allocations})}$$

National Internet Registry (NIR)
NIRs usually maintain an address space for individual countries. However, they are currently used only in Asia. In case of APNIC, they work under the RIR and perform activities for individual countries on behalf of the RIR. [2]

Local Internet Registry (LIR)
LIRs usually allocate an address space to EUs or ISPs. A LIR is usually a concrete ISP, which allocates the address space to its EUs as required. Address blocks should be at least of length `/48`. In some cases, it is however possible to allocate even smaller address block. Nevertheless, it is not common, as it is simpler to allocate a larger address block than recalculate the address space in case of later requirements. [16]

2.5.3 Endpoint address allocation

In protocol IPv6, automatic device configuration is an important feature, which can configure an IP address without user intervention. Today, we recognize two approaches to setting an IP address. The first one is stateful auto configuration, which is based on DHCP, and the second one is stateless auto configuration, which uses a new approach that does not need to communicate with other devices.

Stateful auto configuration
Stateful auto configuration uses DHCPv6, thus it can require user intervention in some cases. A DHCPv6 server stores the list of devices and status information in order to figure out the availability of each address. The DHCPv6 server sends an IP address, Gateway address, prefix length, reservation period, and DNS servers.

DHCPv6 currently has three allocation modes:

- Dynamic – a client gets an IP address for a certain period, thus it has a specified time of validity. However, the client can request IP address renovation at any time.
- Automatic – an IP address is allocated from a scope specified by an administrator. This type of allocation does not include the reservation period and is permanent.
- Manual – a client specifies its IP address and informs the DHCPv6 server.

Stateless auto configuration
Stateless auto configuration (SLAAC) is a new way of IP address allocation that only requires the `/64` prefix of an address and the remaining 64 bits are filled using process EUI-64 (Figure 2.3). This process uses the device's MAC address, which is divided by two octets filled by value FFFE. Finally, the seventh bit is set to one, stating that the IP address comes from SLAAC. [5]

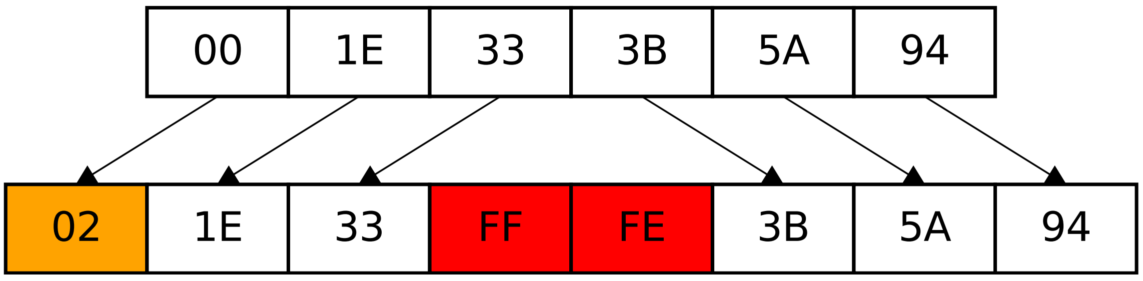

Figure 2.3: MAC address conversion to EUI-64 [15].

2.6 Reasons for IPv6 prefix set generation

Due to the size of an IPv6 address and the protocol's address space, it is necessary to pay much more attention to processing speed. It is also substantial to develop faster, more efficient routing algorithms and new infrastructure. These reasons are obvious especially when we use current routing algorithms in the IPv6 environment. While traversing through such a large address space, the efficiency of the IPv4 routing algorithms is reduced. With the increasing number of IPv6 addresses stored in routing tables, a necessity for more effective routing algorithms will increase. [20]

This leads us to the fact that there are currently not enough allocated IPv6 addresses, which would allow for testing the routing algorithms and infrastructure.

Currently, most allocated prefixes are of length `/32` or `/48`. However, if we consider the allocation policies of registrars, it is possible that the current state will not correspond to the reality in the future. In the future, the majority of allocated prefixes will probably be of the end length of `/64`.

Another problem is that only a small number of organizations publishes their routing tables for research. Moreover, even those routing tables that are published might not correspond to reality, since they might not be a sample large enough.

Chapter 3

Existing IPv6 prefix set generators

Even though there are not many IPv6 prefix set generators, we will mention at least those, which description is available. This chapter describes how each generator works.

3.1 V6Gene

This generator [20] is based on the simulation of address block allocation. It generates new prefixes from trie in order to simulate the allocation performed by all types of registrar. The program works in three phases and the process flow of each phase is shown in Figure 3.1, which is further described below. The flow diagram is divided into three sections. The red section represents initiation, the green section represents generation and the blue section represents outputting.

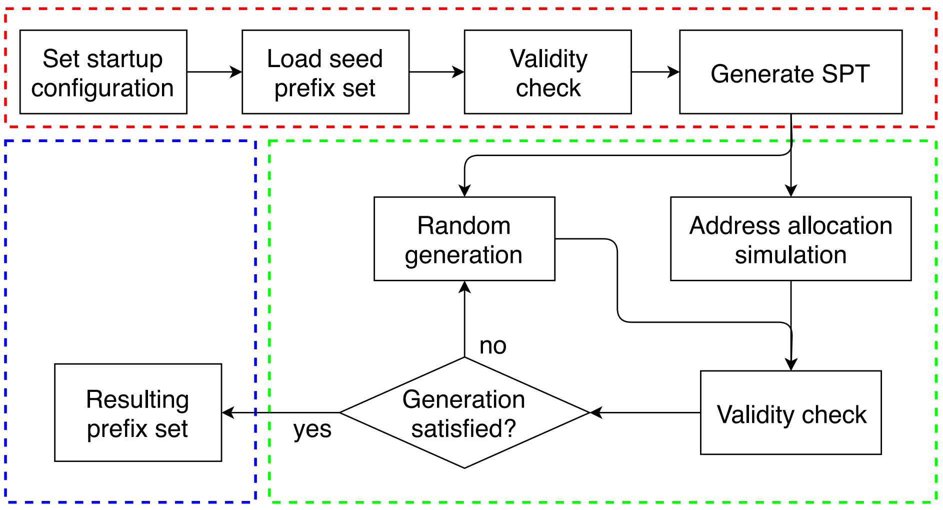

Figure 3.1: Process flow of V6Gene

1. Initiation – In this phase the generator initializes all information based on an input configuration and loads an input prefix set. Then it makes a validity check on the input prefix set and removes duplicities. The validity check is done because of the possibility of multiple prefix sets merged from different routing tables or different paths to destination, which could lead to duplicities. Finally, it generates an initial trie, also called Seed Prefix Trie (SPT).

2. Generating – This phase consists of two parallel processes. The first process simulates address allocation from LIR. This is based on traversing the SPT. Whenever there is a prefix list, the generator starts generating a specific number of prefixes, based on prefix length distribution and prefix levels. All prefixes will start with the same initial prefix. The second process is random generation. It simulates the allocation of new, in the original prefix set not present, LIRs. This stage starts by generating the LIR prefixes and then individual prefixes for each ISP. When this step is done, the generator checks duplicities and IPv6 address integrity. This can lead to the deletion of some addresses in which case the generator starts the generating cycle again until the amount of generated prefixes matches specified numbers.

3. Outputting – This phase outputs the final prefix set in a valid format.

3.2 Non-random Generator

The Non-random Generator [19] expects similarities between the structure of IPv4 and IPv6 routing tables. For this reason, it tries to generate IPv6 prefix sets based on similarities with existing IPv4 prefix sets. The key properties, according to this generator, are the prefix length, the prefix value and the size of a routing table. The generator generates new IPv6 prefixes in two steps, which are further described below. The first step, which reflects prefix length distribution is highlighted in red. The second step, which reflects the prefix value is highlighted in blue.

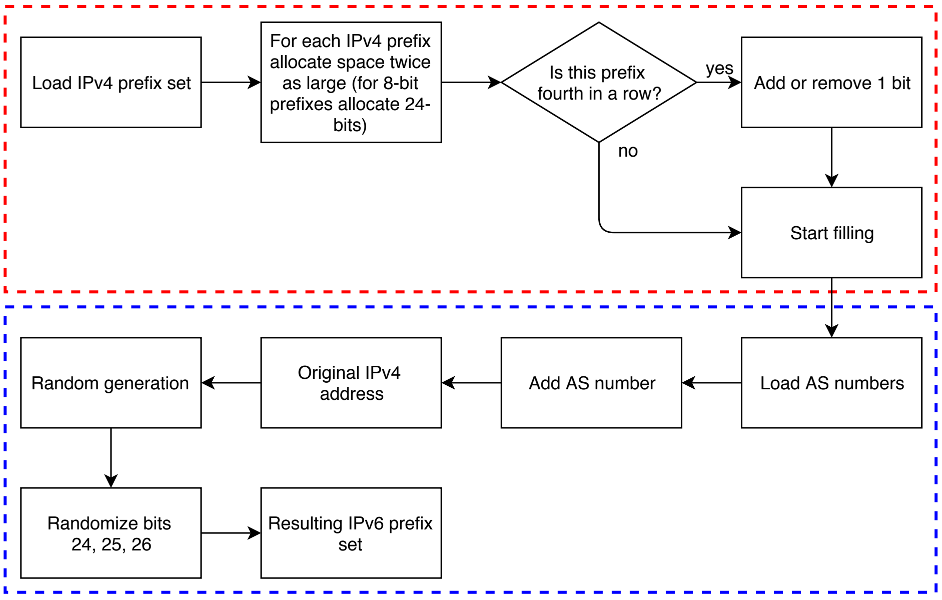
Load IPv4 prefix set
For each IPv4 prefix allocate space twice as large (for 8-bit prefixes allocate 24-bits)
Is this prefix fourth in a row?
yes
Add or remove 1 bit
no
Start filling
Load AS numbers
Add AS number
Original IPv4 address
Random generation
Randomize bits 24, 25, 26
Resulting IPv6 prefix set

Figure 3.2: Process flow of Non-Random Generator

1. Prefix length distribution – There is an assumption that IPv6 prefixes should be twice as long as IPv4 prefixes. However, this will generate only even prefixes. In real IPv4 prefix sets, it is observed that for each three even prefixes there is one odd. The Non-random Generator achieves this by adding or removing one bit from every fourth prefix. Due to the address space allocation policies, the generator also performs recalculation of 8-bit IPv4 prefixes to 24-bit IPv6 prefixes.

2. The prefix value – The generator uses the number of Autonomous Systems (AS) together with the IPv4 prefix because it is a suitable unique combination of 16-bit number of AS together with the 32-bit IPv4 address. Finally, the generator fills the remaining length of the prefix by random bits. Due to the structure of IPv4 prefix sets, it is also ideal for appropriate bit distribution to generate bits between positions 24 and 26 randomly, which should increase address space efficiency.

3.3 IPv6 Table Generator

In contrast with the previous generator [13], this one focuses on generation based on real IPv6 prefix sets, which the generator further expands. It notices that there are not many similarities between the real-life allocation of IPv6 and IPv4 addresses. Instead, the generator uses static data based on real IPv6 prefix sets and generates the required number of prefixes using probability. This means that the generated prefix set should correspond to the original prefix set. The generation consists of two phases. The first phase aims at the generation of new prefixes and the second phase aims at creating the resulting prefix set. Figure 3.3 shows the process flow of this generator. The first phase is highlighted in red and the second phase is highlighted in blue. The generator is further described below.

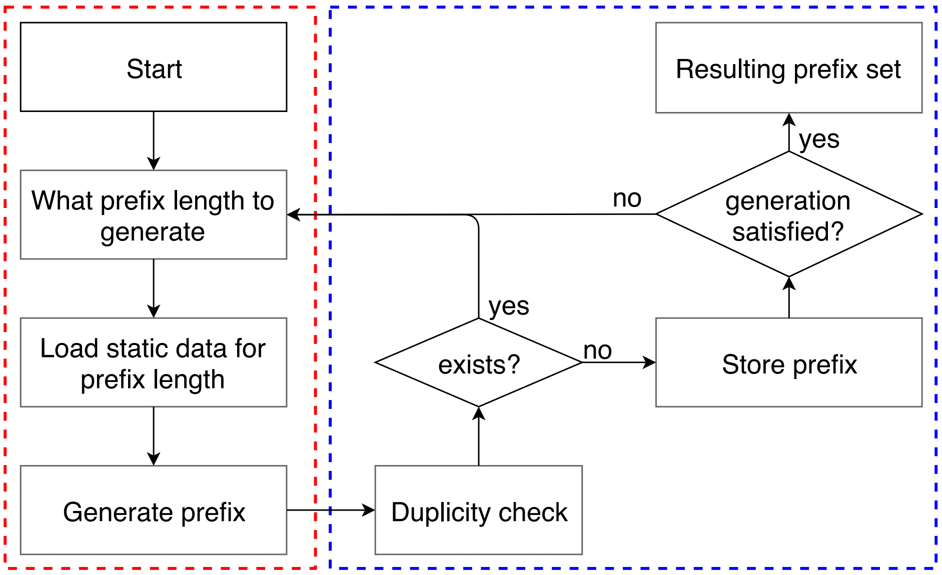

Figure 3.3: Process flow of IPv6 Table Generator

1. The generator uses probability based on a pseudo-random number in interval [0; 100) to determine the length of a generated prefix. Next, it decides what length to generate, loads the static data, which correspond to the specified prefix length, and generates the prefix based on pseudo-random numbers.

2. The second stage checks duplicities. If the prefix was not yet stored, the algorithm stores the prefix and continues to the next one. If the prefix already exists, the algorithm discards the prefix and restarts generation. Generation finishes when the number of prefixes to be generated matches the amount of stored unique prefixes.

Chapter 4

Proposal of usable properties

The beginning of this chapter deals with IPv6 address and its properties, including tries. The second part proposes additional tests specific for generators.

4.1 Basic properties of prefix sets

Prefix sets contain several similarities between each other. This is mostly due to the way addresses are allocated (Chapter 2). Even though IPv6 addresses are noted in hexadecimal, in computer world they are stored in binary. This means that we should try to understand an address on the binary level and find similarities. The main similarities that we can notice are in prefix length distribution and bit value distribution. [19] [13]

4.1.1 Prefix length distribution

IPv6 prefix sets consist of prefixes of various lengths. According to current address allocation principles mentioned in Chapter 2, we can expect mainly prefixes of lengths `/32`, `/48` and `/64` to be allocated. By using prefix generators, we are trying to get larger prefix sets than those currently available. Therefore, we need to figure out a way of generating prefixes that are as close to reality as possible. The prefix length distribution can be therefore used as a simple comparison between generated prefixes and those we can expect in future. Talking about the future, we should take into account that with an increasing number of addresses the percentage representation of `/32` addresses will be lower and will transfer to `/48` and later to `/64` prefixes.

4.1.2 Bit value distribution

Each bit of IPv6 prefix can get one of two states. This is something we can study by calculating the probabilities of ones and zeros at each bit. While looking at real prefix sets, we can notice that the more sample bits of prefixes we get, the more balanced distribution of zeros and ones we see. This is not currently true mainly for the first three bits because there is only one `2000::/3` unicast address block released. Ignoring this fact, we can expect that bits with most samples would approach balanced distribution while bits with less samples would be extreme in terms of distribution.

4.2 Properties of trie

A prefix tree, also known as trie, is the basic data structure for storing a prefix set used in the majority of routing tables. Trie is a binary tree meaning that there are at most two children leading from one node. [20] [14]

In trie, prefixes are represented by a path to a node, which is different from binary tree, where the data are stored inside of nodes. This means that trie cannot self-balance itself without losing a stored value.

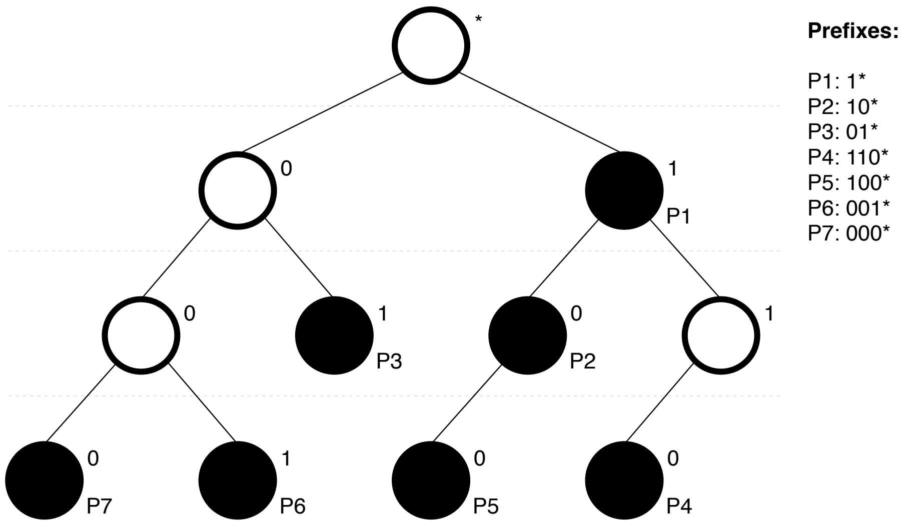

Figure 4.1: Example of a trie

Figure 4.1 shows the structure of trie. There are seven prefixes stored inside of the trie, which are noted in the table. The root node itself does not represent any value and is located at level 0. To simplify the description of storing values into trie we shall assume that right edge represents binary one and the left edge represents binary zero.

In order to store prefix P1, we have to go from the root on the right edge to store leaf P1. To store P2 we will go the same path as for P1 and continue on the left edge creating a new leaf and making P1 a branch node. This way we can create trie shown in Figure 4.1.

Prefix lookup in a trie works similarly. Let us take an example prefix `2001::1`, which is also currently the only unicast address range released. Translated to binary it starts as `0010*`. A search algorithm starts at the root node and descends on the left edge. From this position, it continues once again via the left edge and finally by the right edge. Since this is the best match we can get for said address, the search algorithm stops.

There are several properties of tries that can be observed. As we can see, there are some nested prefixes. Trie, as stated above, cannot be balanced and there are different probabilities of branching at different levels.

4.2.1 Prefix nesting

One of the properties of prefixes is nesting. We can observe this when allocating prefixes from regional registry's `/32` prefix to local registry's `/48` prefix. LIR's prefix is therefore nested into RIR's prefix. Prefix nesting thus occurs when several prefixes are stored on the same path of trie. [4]

In Figure 4.1 we can see several nested prefixes: `1*`, `10*`, `100*`, and `110*`. In other words, prefix 1* contains other prefixes and generalizes them, hence we say this prefix is shorter and matches to more addresses than any other prefix nested inside it.

For testing purposes, we will take into consideration only nesting values of leaf nodes. The path to prefix `100*` in the example, which is a leaf node, has prefix nesting of 3, while prefix `110*` has nesting of 2.

4.2.2 Branching of trie

Branching occurs when a trie node has more than one child. Trie is usually not balanced, thus branching of trie is not guaranteed. We can however talk about branching probability at different levels.

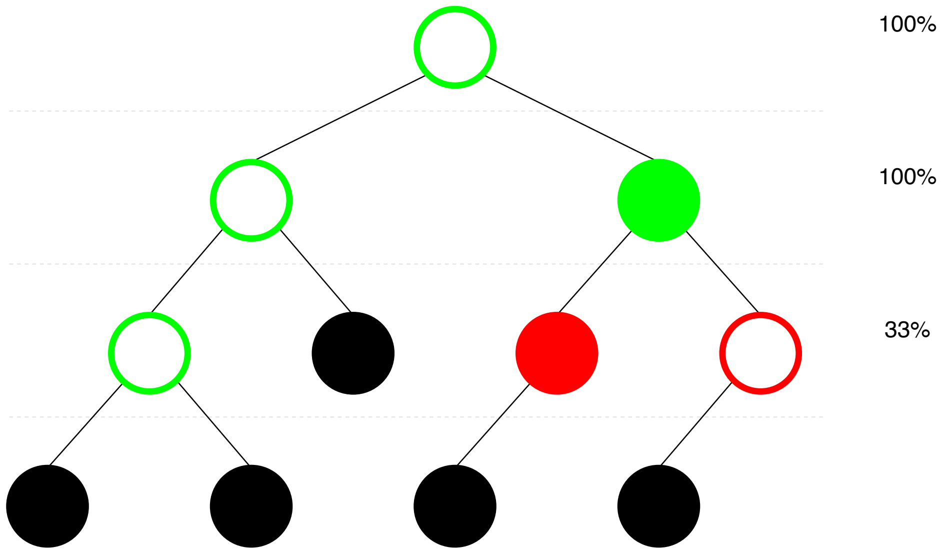

Figure 4.2: Probability of branching

To get an idea about structure of said trie, we need to know how often it can branch and more precisely at what levels the branching occurs most. This is something we cannot notice while talking about probability of bit positions described above.

In the example in Figure 4.2 the red nodes represent non-branching nodes and the green ones represent branching nodes. We can see that the closer to the root the more we can expect branching. This is something that differs among prefix sets and can affect the performance of search algorithms.

To calculate branching probability, we take into consideration only nodes with at least one successor. For example, at level two in Figure 4.2 there are three non-leaf nodes and one leaf. Since we are calculating branching probability for 2-children nodes, we use Equation 4.1

$$branching = \frac{\text{2-children nodes}}{\text{all non-leaf nodes}} * 100 \tag{4.1}$$

Hence, the example at level two would have branching probability of 33% for 2-children nodes.

4.2.3 Skew of trie

It is desired to have balanced trees for optimal searching in binary trees. While a classic binary tree can balance itself, in trie a path represents a prefix, thus cannot be rebalanced. However, in a trie we can calculate the weight of each subtree to calculate the skew of branching nodes. [14]

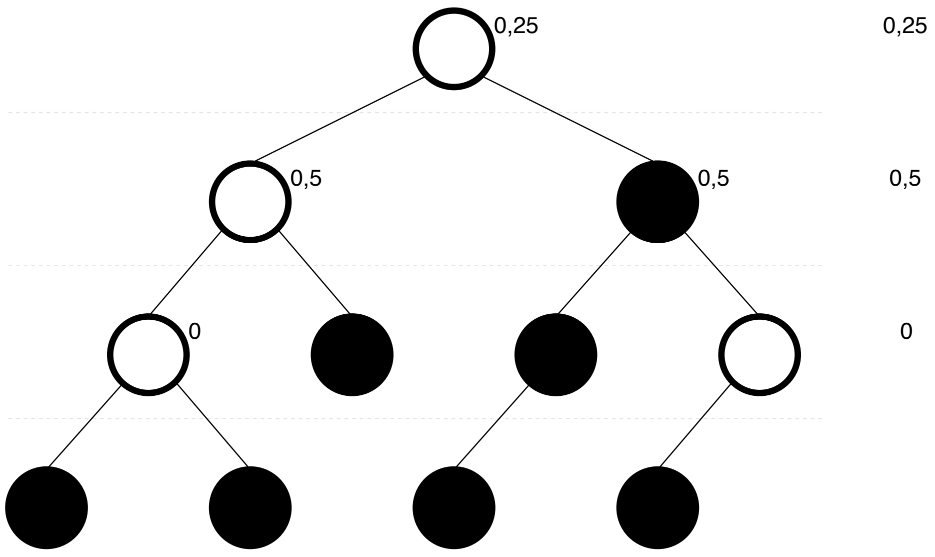

Figure 4.3: Calculated skew for nodes and levels

We can only calculate skew for branching nodes and generalize the skew by level. To calculate the skew of the root node in Figure 4.3, we can use Equation 4.2

$$skew = 1 - \frac{\text{weight of lighter subtree}}{\text{weight of heavier subtree}} \tag{4.2}$$

In this equation lighter subtree represents the subtree with less prefix nodes in contrast with a heavier subtree, which contains more prefix nodes. The result would be:

$$skew = 1 - \frac{3}{4} = 0,25$$

For skew at selected level we then sum all skews and divide it by the number of branching nodes.

4.3 Properties of prefix generators

While other properties focus mainly on IPv6 representation, there are also another aspects of generating prefixes. Mainly the number of duplicates in generated samples, as there are no duplicates in forwarding tables. The second aspect are hardware requirements required by the generator for prefix set generation.

4.3.1 Duplicates

Although raw data from routing tables contain duplicates, when we consider forwarding tables where only the best paths are stored, there are no duplicates. Prefix sets that we want to be generated therefore do not need duplicates and we might consider them as redundant data. It should also be faster to proceed with tests if we remove all unnecessary data.

4.3.2 Hardware requirements

Sometimes we might want to generate data in timely manner due to computer time. Other times we are generating large sets of data without much RAM. For these reasons, it is also important to know other aspects of said generators than only the generated data. We propose to gather RAM consumption, disk R/W operations and CPU time as a simple way to determine the positives and negatives of each generator hardware requirements. These data can also be later compared between different generators or runs to gather an average.

Chapter 5

Implementation of prefix set comparison

This chapter consists of two subchapters. The first one is proposed way, which proposes the ways of comparison of IPv6 prefix sets and the structure of our script for comparison. The second subchapter consists of the implementation of the script and the possible ways of run.

5.1 Proposed way

This subchapter aims at the comparison of generators in contrast with reality.

5.1.1 Optimal testing

As described earlier, generators should follow registrar policies. This means that we can expect similarities to some extent between generated sets and reality. Even though this is true, reality might be slightly different in comparison to registrar policies, but still close. This could be for example due to a local anomaly. However, even in this case we should still aim to get a sample as close to our observed reality as possible.

To test this, we cannot predict the future. Instead, we can use prefix sets from the past and generate new prefix sets which we can compare with current prefix sets. However, for this we need data from routing tables that go far enough in the past, because we need samples from the same routing table to compare such samples. Using this technique, we can study the difference between current original set and its generated version from the past.

5.1.2 Operation of script

The script can be divided into three sections of operation shown in Figure 5.1.

The first section is highlighted in red and focuses on gathering statistic data about prefix sets, which either already exist or should be generated by a generator. If we want to first generate a new prefix set via this script, it will start by capturing the hardware resources taken by the generator. After that we have to decide, which group of tests we want to test our prefix set on and decide whether or not we also want graphs for each test.

The second section consists of operations over existing results. In this case, the script does not need to recalculate each stat from the original prefix set. Instead, it uses already calculated results, which are stored. These data can be used later for further manipulation, such as graphs. We can also use these data to calculate RMSE between samples or merge multiple samples in order to increase comparison accuracy. In case of RMSE, we can also print graphs.

Finally, the last section consists of erasing all data gathered in order to start a new series of tests.

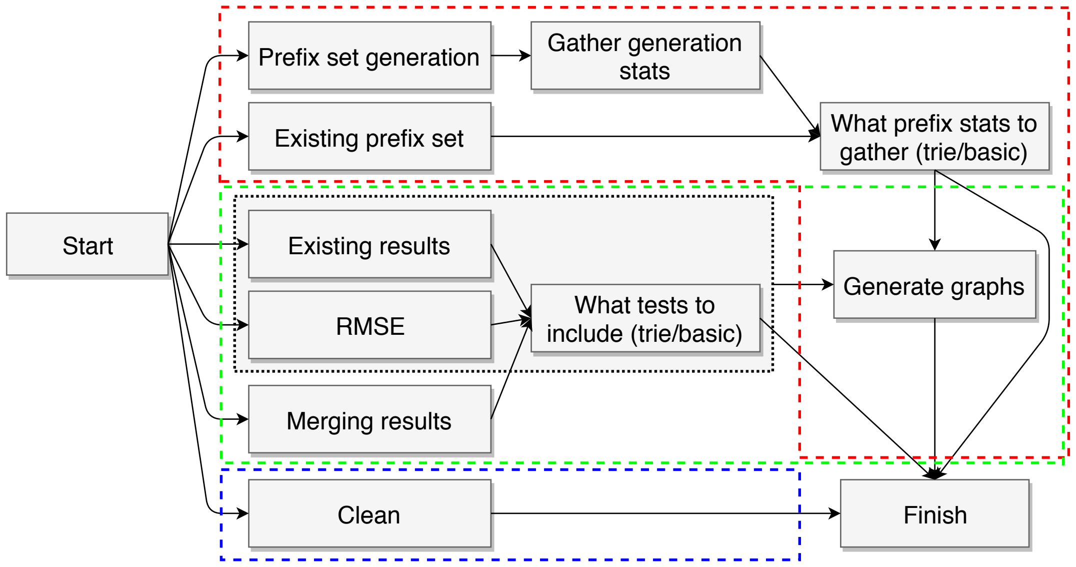

Figure 5.1: Script flow chart

5.1.3 Interpreting results

If we consider the size of routing tables, it is obvious that we do not need to work with exact numbers for each bit. Instead, we need to know the percentage representation throughout the sample set. This way we can also compare smaller sets with larger ones. This works for basic results of sets; however, if we want to compare such samples between each other or in comparison to the original prefix set, it is better to use Root Mean Square Error (RMSE). The RMSE is shown in Equation 5.1, where $\hat{y}$ represents the original prefix set and y_t represents the generated prefix set for n samples.

RMSE shows us the average difference between the generated set and the original. In other words, it measures the error between these sets. We can even merge multiple generated sets and compare these to the original sample, which better corresponds to the generator. [8]

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n} (\hat{y} - y_t)^2}{n}} \tag{5.1}$$

5.2 Implementation

Each test, except for prefix length distribution, requires translating an IPv6 address into binary. This means that we should first translate our addresses into binary for simpler use.

There were several approached problems during implementation. The first was to find an effective way of working with a large number of IPv6 prefixes. The original idea was to create separate scripts for each test described in Chapter 4 but mainly due to the need to use the same data in each script, we decided to create one larger script, which partly divides the tests and reduces overall time and allocations. Therefore, we have divided two basic tests into separate functions, while the trie tests are calculated simultaneously. Another problem occurred when we exported prefix sets from routing tables as there were many duplicates. Followed by the need to translate each IPv6 prefix from hexadecimal to binary for every script. It was decided to first translate all prefixes into binary and then remove duplicities to remove any duplicates hidden in different IPv6 representation and store them back into hexadecimal for better readability.

Based on Chapter 2 where we explain prefix allocation from registrars, it was expected that only prefixes of length 64 and shorter would get into routing tables. While looking at real prefix sets, we can however see this is not true. Based on the total number of prefixes in a set, there can be up to several hundred prefixes longer than 64. These prefixes might not be significant for a set of one million prefixes. However, it is still a property that might be missing in generators that are strictly based on allocation policies of generators. Due to this reason, we have decided to calculate the tests to length of 128. However, in graphs these will not be taken into account.

As noted earlier, the first idea was to create several independent scripts that would perform tests on generated prefix sets. At first, prefix length distribution and probability of zero were written this way. However, it was soon obvious that creating a series of independent tests would take a lot of unnecessary resources. Since the first two basic tests are not much time consuming, they were included into a larger script.

Tests on trie were another challenge. Since we could not use a simple binary tree, because in tries a path represents a value, we had to create our own tree that would allow us to fill it the way we needed. In this scenario, it was soon noticeable that we have decided to go the right way of trying to generalize the trie tests as much as possible because otherwise it would take up to four times longer to get all the information for each test. Eventually, it was necessary to pass the trie twice in order to retrieve all trie information about the prefix set. It was necessary to go at least once from the root to leaf nodes and then recursively level by level from bottom of trie to the root.

5.2.1 Running script

The final script has several ways of operation. In order to further understand each switch, we can refer to Figure 5.2, which shows switches described in this section in the flow diagram of the script. There are two ways of initializing a prefix set; either by giving the script an already existing prefix set by the `-input="sample"` switch, or by starting a generator by the script using the `-gen="run sample generator"` switch to first generate a new prefix set. In order to gather hardware requirements information about prefix set generation, it is necessary to include the `-r` switch. In order to start tests on a specified prefix set, it is necessary to specify which prefix set properties should be tested. For basic tests containing the probability of zero, prefix length distribution, and the number of duplicates in a sample use the `-b` switch and for tests on trie add switch `-t`. To specify the name of a said test, it is possible to add a name using switch `-name="sample"`.

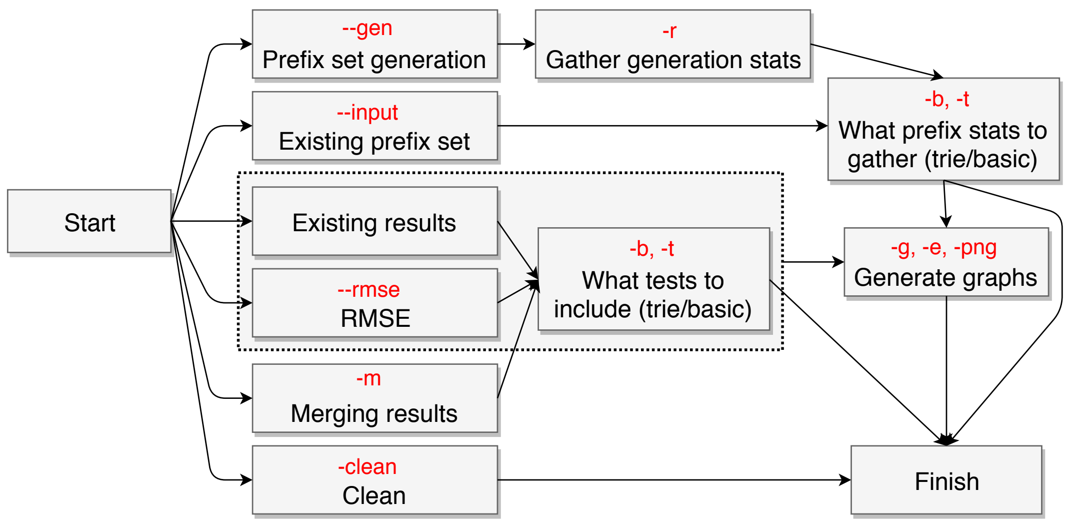

Figure 5.2: Switches in the script

It is currently not possible to make the script run a generator multiple times. Instead, to get average samples, it is necessary to start the generator manually multiple times either independently or via the script. It is then possible to merge multiple tests by specifying their IDs using the `-m="id,id"` switch. Such a sample can be now used in RMSE.

To calculate root mean square error, it is necessary to have two finished test samples, which were first calculated for the same test ranges (either basic/trie or both). It is then possible to calculate RMSE using switch `-rmse="source_id, generated_id"`, where `source_id` represents the original prefix set and `generated_id` represents either one prefix set or a series of prefix sets that were first merged using the `-m` switch. After that it is necessary to specify whether to calculate basic or trie tests using `-b` and `-t` switches.

We can generate graphs right after finishing tests on a prefix set. To do this, we need to specify one of these switches `-g`, `-g="id,id"` for classic graphs, where the first switch takes the last two tests and the second one takes either one or two specified tests. Another possible switches are `-e`, `-e="id,id"`. These generate graphs for tests with already calculated RMSE. Finally, by default graphs are generated in the `.eps` format. It is also possible to generate graphs in the `.png` format by specifying switch `-png`.

5.2.2 Solving the prefix set comparison

The script starts by checking used switches, whether or not they are compatible with each other. After that it sets run ID, which will be later used to store the results and as a reference in case of use in graphs.

As noted earlier, there are two ways of initializing a prefix set, either by the `-input` or `-gen` switch. If we decide to use the `-gen` switch, then the script starts the generator specified in the command line and stores information about its run so it can later be used for comparison between different generators. After that, the script captures the output of the generator, which will be used as a generated prefix set.

The provided prefix set is then translated from IPv6 notation to binary and cut by the prefix length. After this transformation, it is now easy to get the number of duplicates and remove them as no prefix is hidden behind different IPv6 addresses with the same prefix length. Such prefix set can now be processed faster for each test since there are no duplicates anymore.

As stated above, the probability of zero and prefix length distribution are calculated separately. These tests start with an already filtered prefix set from the previous step and create output files containing statistics about said properties. Trie tests are similar. At first, the filtered prefix set is loaded into a binary tree-like structure. With filled trie, all information for skew, branching and nesting are now calculated through two passes via the trie. The first from the root node to bottom and the second vice versa. Finally, all results are stored similarly as in the basic tests.

Root mean square error requires a reference sample and a sample to which we want to compare. First of all, it needs to know which tests to compare. This is set by the `-b` and `-t` switches. Then it compares the original sample with one or multiple samples based on the state of the ID. The ID can represent multiple samples if they were first merged by the `-m` switch. If the original sample contained less prefixes, it is possible to vary more at lower bits. These will be temporarily filled with zeros for comparison and taken into account only if they occur in the generated sample.

Currently, graphs are done via gnuplot. By default, it is possible to print one sample in a graph or compare two samples with each other. If we need to compare more than two samples in one graph, it is necessary to alter a gnuplot script.

Chapter 6

Comparison of existing generators

Even though in Chapter 3 we were describing three generators, currently there are only two of them available. In this chapter, we will compare all these generators against a real prefix set using the current-past window described in Chapter 5. The structure of Section 6.1 corresponds to the one of Chapter 4. Section 6.2 summarizes this chapter.

6.1 Comparison with real prefix set

In this section, we will compare the prefix set generation of both generators based on a prefix set from year 2013 consisting of approximately 13 thousand unique values against a prefix set from year 2018 consisting of almost 50 thousand unique values. With this, we aim to show how accurate these generators are at predicting the course of address allocation.

In order to have comparable prefix sets, we aim to generate prefix sets with approximately 50 thousand prefixes. However, this is not possible for the nonRandom generator, which generates as many prefixes as the input IPv4 prefix set consists of. To compensate this flaw, we run a sampling script over the IPv4 prefix set, which randomly selects 50 thousand prefixes that are then used. This is done multiple times in order to maximize the accuracy.

Each graph shown in this section is in the “RMSE format“. This means that the vertical axis represents the Root Mean Square Error calculated from multiple runs of these generators for different bits and trie levels shown on the horizontal axis.

6.1.1 Basic properties of prefix sets

This section evaluates the two key properties that both generators use while generating their output prefix sets. These properties were described in detail in Chapter 4.

Prefix length distribution

As we can see in Figure 6.1, both generators have similar results with respect to prefix length distribution. However, the error for both generators is based on different data composition.

The NonRandom generator starts from an IPv4 prefix set and doubles the size of every prefix as described in Chapter 2. The address space of protocol IPv4 is already close to exhaustion, which means that the prefix length distribution is skewed towards longer prefixes. This is what we see in Figure 6.1, where the error for prefix length `/32` is caused by more existing prefixes longer than prefix `/32` (i.e., longer than prefix `/16` in the original

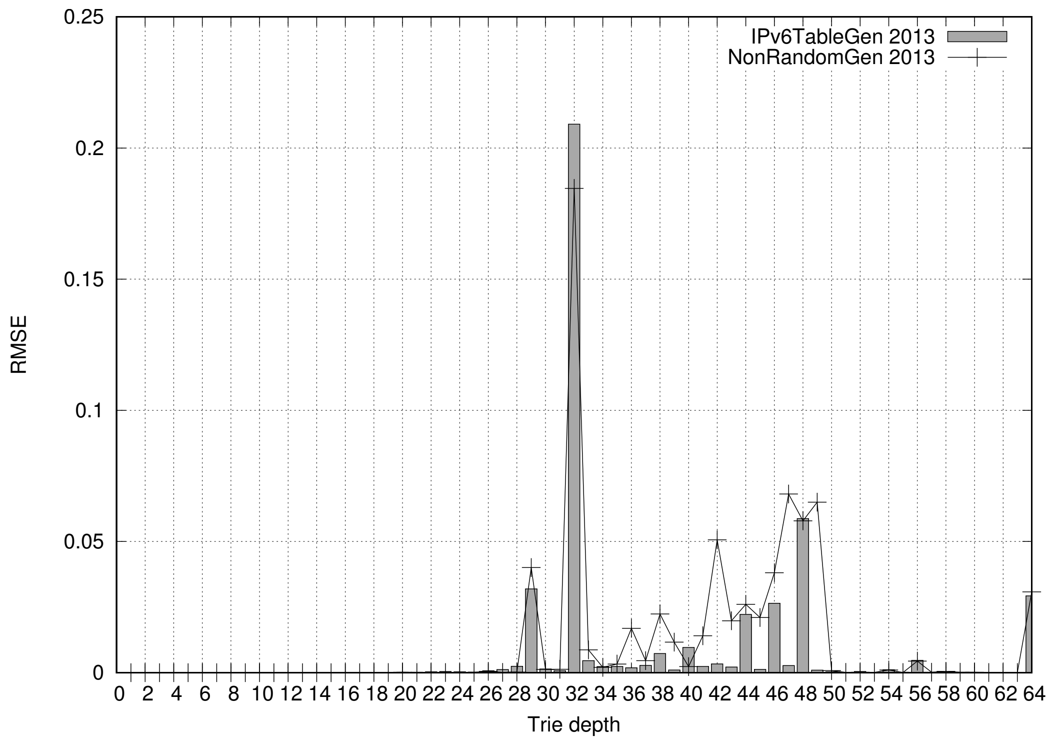

Figure 6.1: Comparison of generated prefix sets on prefix length distribution using RMSE

IPv4 prefix set) in comparison to the target prefix set. In Figure 6.2 we can see the prefix length distribution of an IPv6 prefix set generated by the NonRandom generator. Considering prefix length doubling by the NonRandom generator, the prefix length distribution corresponds to that of an IPv4 prefix set. This means there are almost no prefixes longer than /48 in the generated IPv6 prefix set and the number of prefixes of length /32 is minimal. According to current allocation policies, this could reflect the near future because of a shift from /32 prefixes to /48 prefixes (see the prefix set from year 2013, represented by the output of the IPv6Table generator, and the original prefix set from year 2018 in Figure 6.2). However, the generator does not count with border prefix lengths of registrar ranges described in Chapter 2 such as RIR range between prefix /32 and /48, which results in larger error for near prefix lengths.

The IPv6Table generator uses an IPv6 prefix set and generates the output prefix set using the same prefix length distribution. As we can see in Figure 6.1, the error for prefix length /32 is almost the same as for the NonRandom generator. However, in Figure 6.2 we can in fact see that the error has different origin. This means that with larger prefix sets the error would most likely increase.

In Figure 6.2 we can notice a difference between the prefix set generated by the IPv6Table generator according to the prefix set from year 2013 and the original prefix set from year 2018, which corresponds to the current allocation policies. There is a noticeable shift from the prefix length of /32 to prefixes between length /32 and /48 that we can describe as the first stage of IPv6 adaptation. In the end, this stage should end similarly as the prefix set generated by the NonRandom generator. Eventually, in the second stage the length of the majority of prefixes will shift to /64. We can already notice the beginning of this stage in the original prefix set from year 2018.

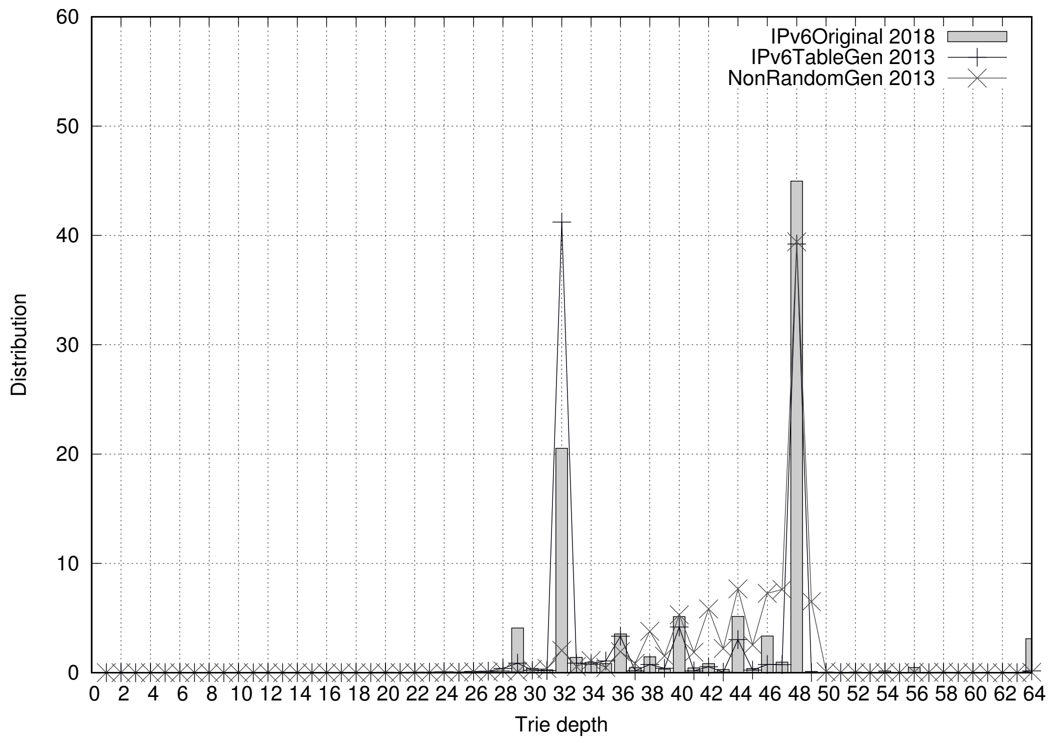

Figure 6.2: Comparison of prefix length distribution of a real IPv6 prefix set and two IPv6 prefix sets that were generated according to prefix sets from year 2013

If we consider the current allocation policies and what we have observed in the previous paragraph, the V6Gene generator should potentially best reflect the course of future IPv6 address allocations. Nevertheless, its implementation is currently not publicly available.

Bit value distribution

Figure 6.3 shows the bit value distribution at each bit. If we consider the bit value as another important aspect for the NonRandom generator, we might assume its generation process to be quite random at some bits. However, the problem is due to the probability of zero of the original IPv4 prefix set and AS numbers, which are different when compared to IPv6 prefix sets. On the other hand, the IPv6Table generator shows a mild error from the original prefix set.

Prefix sets generated according to a prefix set from year 2018 almost copy those based on the original prefix set from 2013. The prefix set from the IPv6Table generator has smaller error while the prefix set from the NonRandom generator remains the same. However, in case of larger prefix sets this error might change as the bit value distribution could converge to fifty percent in the future, as described in Chapter 4.

6.1.2 Properties of trie

This section offers another view on the prefix set generation. This time, we will consider the way data are organized in trie. Trie tests might better reflect the inherent structure of data in larger samples where trie is large. These tests are further described in Chapter 4.

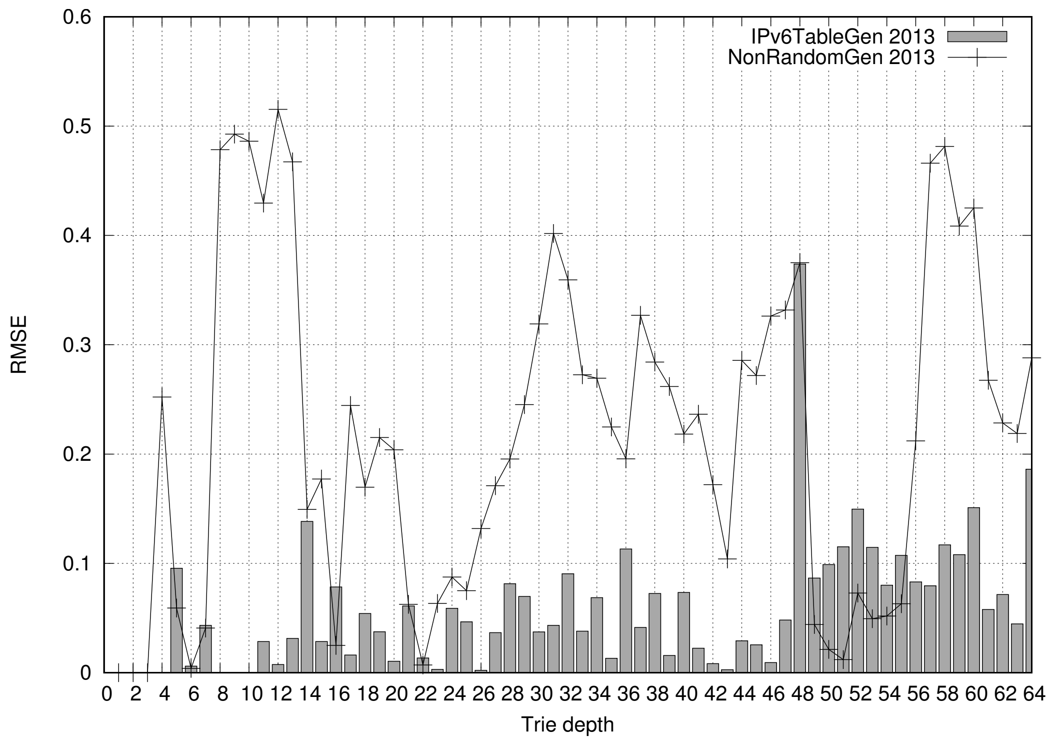

Figure 6.3: Comparison of generated prefix sets on bit value distribution using RMSE

Prefix nesting

According to Figure 6.4, both generators almost exactly correspond to the prefix nesting of the target prefix set. This is partly due to the size of the prefix set. More nesting occurs when there are many prefixes in the set.

Branching of trie

Figure 6.5 depicts branching on individual levels of trie. In this case, difference from the original prefix set occurs mostly at the first levels of trie. While the NonRandom generator has significantly larger branching at these levels, the original prefix set does not branch at all at these levels. In case of the IPv6Table generator the prefix set sometimes branches also in the root node. Considering the current allocation policies, which allocate only one unicast prefix of length three, the V6Gene generator should not branch prematurely since its main goal is to expand already existing prefixes in order to simulate allocation from individual registrars.

Skew of trie

According to skew in Figure 6.6, both generators have similar results. Although the Non-Random generator has slightly larger skew at the first bits, it is not significant. However, skew might change in larger prefix sets where trie is more populated.

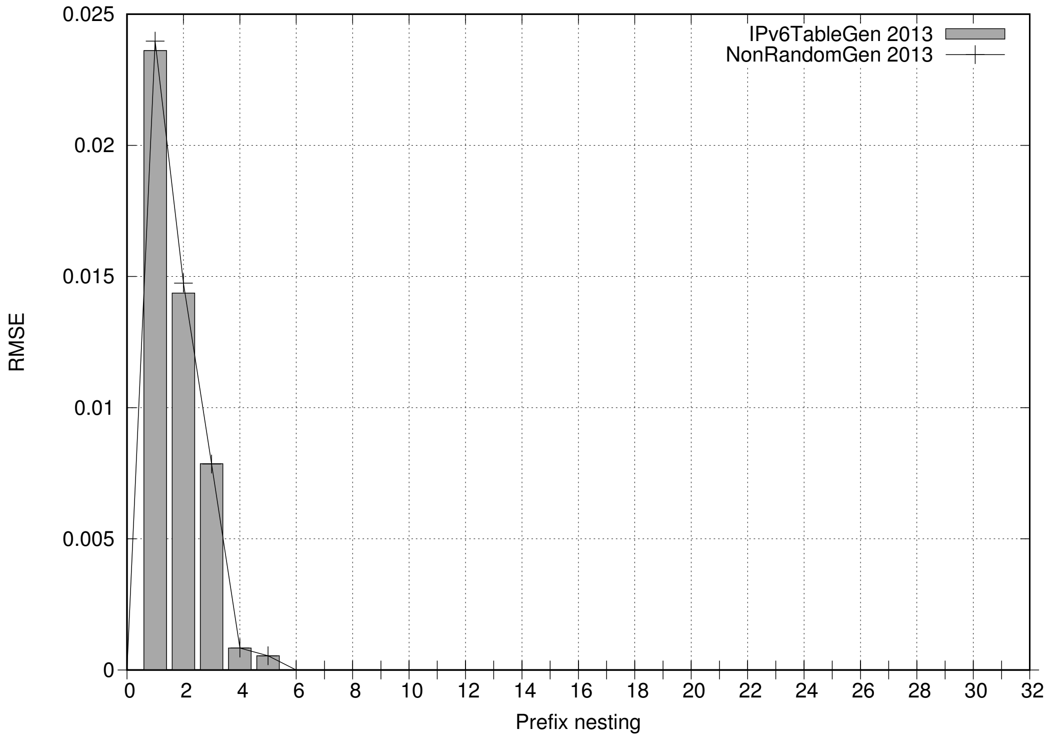

Figure 6.4: Comparison of generated prefix sets on prefix nesting using RMSE

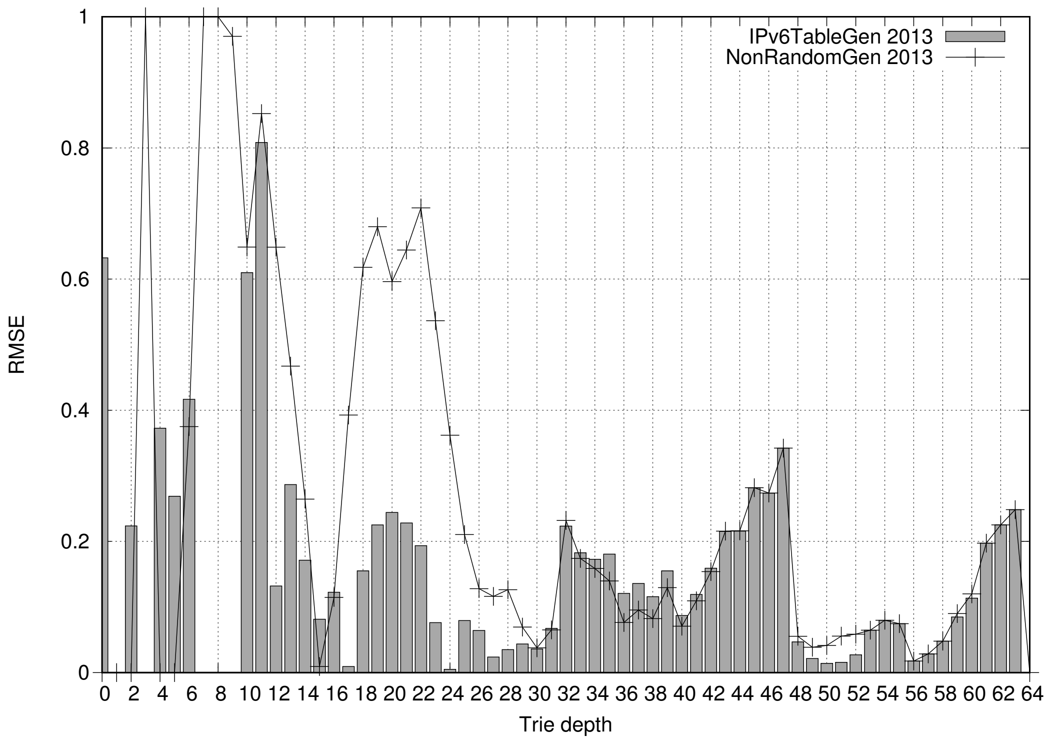

Figure 6.5: Comparison of generated prefix sets on branching using RMSE

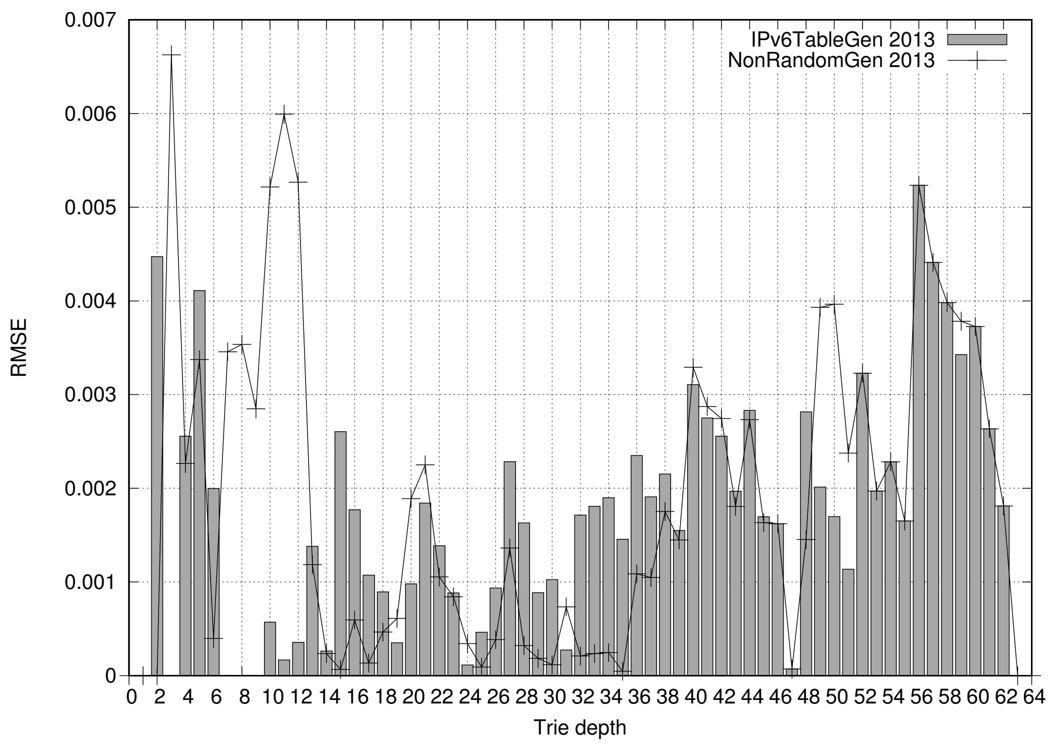

Figure 6.6: Comparison of generated prefix sets on skew using RMSE

Prefix set	**Total number of duplicates**
IPv6Table 2013	0
nonRandom 2013	4

Table 6.1: The number of duplicates

6.1.3 Properties of prefix generators

This section aims to evaluate the effectivity of the generators. These tests are described in more details in Chapter 4.

Duplicates

Table 6.1 depicts the number of duplicates in generated prefix sets. It is also necessary to remind that the NonRandom generator only expands the original IPv4 prefix set. For this reason, the number of duplicates greatly depends on the original prefix set used for prefix set generation. In our case we first remove the duplicates from the original IPv4 prefix set. If the original prefix set comes from a routing table, the generator should first remove duplicates in order to reduce the generation time.

Hardware requirements

In order to better illustrate hardware requirements, we have created the radar graph shown in Figure 6.7, which compares both generators. In this graph, 100 % represents the generator, which has worse results per prefix for specified criteria. In this case, the IPv6Table

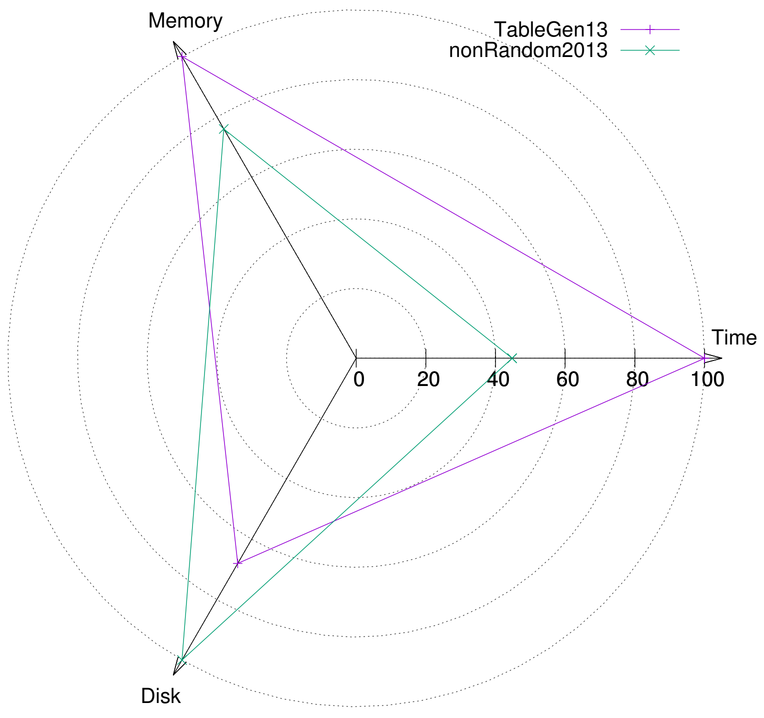

Figure 6.7: Hardware requirements

generator has larger memory and processor time requirements, while the NonRandom generator has larger disk requirements.

6.2 Summary

According to the basic tests, neither generator really takes into account the currently active allocation policies. While the IPv6Table generator simply reflects the current status and will not be accurate as the routing table grows, the NonRandom generator might be accurate for a short period of time in the future. However, this means that we cannot use this generator in order to generate prefix sets larger than already existing IPv4 prefix sets, which is one of the main reasons of IPv6 adoption. On the other hand, the V6Gene generator is proposed to work according to the allocation policies, thus better reflect the future prefix sets.

The trie properties tests have shown that skew and nesting mostly correspond to the target prefix set in this case. Nesting corresponds due to the size of the prefix set and allocation policies, which are almost the same for both IPv4 and IPv6. Skew has a small error due to a balanced prefix layout throughout trie in comparison with the target prefix set. Finally, branching varies greatly even in these small prefix sets. For the NonRandom generator, this is possibly due to the layout of bits in AS numbers and IPv4 addresses. In case of the IPv6Table generator, it could be the property of a used pseudo-random number generator. Theoretically, this might mean that the used pseudo-random number generation better corresponds to the reality of IPv6 prefix sets than generation based on existing IPv4 prefix sets. Nonetheless, since the V6Gene generator generates most of its prefixes from already existing ones, it should have minimal RMSE at the first levels of trie in contrast with both generators.

Based on the properties of prefix set generators, we can say that the IPv6Table generator is more effective in terms of disk usage and does not generate any duplicities. On the other hand, the NonRandom generator requires less processor time per prefix and less memory.

Neither of these two generators correspond with all series of tests without large error. If we want to get a prefix set as close to reality as possible, we might want to implement the V6Gene generator first in order to compare it with other generators. According to the description of all three generators and the results of those two available, the V6Gene generator should have the most accurate results.

Chapter 7

Conclusion

The main contribution of this bachelor's thesis is a script, which implements a series of tests for comparing IPv6 prefix set generators. In order to demonstrate the usability of this script, we have utilized it for comparing the currently available generators.

In Chapter 2, we have described the IPv6 protocol including current allocation policies for new prefixes and addresses in order to better understand the following chapters. In Chapter 3, we have described three IPv6 prefix set generators and presented flow diagrams describing each of them to better understand the generation of new prefixes.

In Chapter 4, we have proposed the set of testing criteria, which target the criteria used by the generators for the generation of new prefixes as well as the effectivity of each generator. These criteria are divided into three groups. The first one comprises basic properties, the second focuses on trie properties, and the final group consists of the performance properties of the generators. In the first section of Chapter 5, we have proposed the way of testing said generators and a script used for such testing. In the second section of Chapter 5, we have summarized the implementation of our script and described the way it operates. Finally, in the last chapter we have successfully compared two available generators using our script in order to determine which one of them is more accurate in terms of prefix set generation. The conclusion based on our tests is that both tested generators contain large error for some tests in comparison to real prefix set as described in the last section of Chapter 6. As observed in Chapter 6, the NonRandom generator might generate almost accurate results for a specific size of prefix set. However, based on our observation and description of the V6Gene generator, it is the only generator out of those described in this thesis that should generate a prefix set that is close to real prefix set.

In the future, it could be useful to further expand the set of tests and experiment with larger prefix sets in order to observe if the results gathered by these tests would be accurate in the future. It is also useful to optimize the script in order to work more effectively. Finally, if the V6Gene generator becomes available, it would be interesting to compare it with the other two generators. In order to determine how accurate were our presumptions described in Chapter 6.

Bibliography

[1] Arkko, J.; Ericsson, E.; Kempf, J.: SEcure Neighbor Discovery. Accessed: 2017-01-25.
Retrieved from: https://tools.ietf.org/html/rfc3971

[2] Asia-Pacific Network Information Centre: National Internet Registries. Accessed: 2016-12-25.
Retrieved from: https://www.apnic.net/about-APNIC/organization/apnics-region/national-internet-registries

[3] Asia-Pacific Network Information Centre: Understanding address management hierarchy. Accessed: 2017-02-01.
Retrieved from: https://www.apnic.net/manage-ip/manage-resources/address-management-objectives/management-hierarchy

[4] Baccala, B.: Nested prefixes. Accessed: 2018-05-14.
Retrieved from: http://www.freesoft.org/CIE/Course/Subnet/9a.htm

[5] Das, K.: Stateless Auto Configuration. Accessed: 2017-01-23.
Retrieved from:
http://ipv6.com/articles/general/Stateless-Auto-Configuration.htm

[6] Durand, A.; Huitema, C.: The Host-Density Ratio for Address Assignment Efficiency: An update on the H ratio. Request for Comments: 3194. November 2001. Accessed: 2017-02-25.
Retrieved from: https://tools.ietf.org/html/rfc3194

[7] Hinden, R.; Deering, S.: IP Version 6 Addressing Architecture. Request for Comments: 4291. February 2006. Accessed: 2017-02-25.
Retrieved from: https://tools.ietf.org/html/rfc4291

[8] Holmes, S.: RMS Error. Accessed: 2018-05-14.
Retrieved from:
http://statweb.stanford.edu/~susan/courses/s60/split/node60.html

[9] Huston, G.; Lord, A.; Smith, P.: IPv6 Address Prefix Reserved for Documentation. Request for Comments: 3849. July 2004. Accessed: 2017-02-25.
Retrieved from: https://tools.ietf.org/html/rfc3849

[10] Internet Assigned Numbers Authority: Number Resources. Accessed: 2017-02-01.
Retrieved from: https://www.iana.org/numbers

[11] Internet Corporation for Assigned Names and Numbers: Internet Assigned Numbers Authority (IANA) Policy For Allocation of IPv6 Blocks to Regional Internet

Registries. February 2012. Accessed: 2016-12-25.
Retrieved from: https://www.icann.org/resources/pages/allocation-ipv6-rirs-2012-02-25-en

[12] IPv6 Now Pty Ltd: Reasons for IPv6. Accessed: 2017-01-25.
Retrieved from: http://ipv6now.com.au/primers/IPv6Reasons.php

[13] Lorenc, M.: *Generator of IPv6 tables*. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. 2013. Accessed: 2016-12-25.
Retrieved from: http://www.fit.vutbr.cz/study/DP/BP.php?id=15157

[14] Matoušek, J.; Antichi, G.; Lučanský, A.; et al.: ClassBench-ng: Recasting ClassBench after a Decade of Network Evolution. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. May 2017. ISBN 9781509063864. pp. 204–216. doi:10.1109/ANCS.2017.33.

[15] Mro: Ipv6 eui64. Accessed: 2018-05-14.
Retrieved from: https://en.wikipedia.org/wiki/File:Ipv6_eui64.svg

[16] Réseaux IP Européens Network Coordination Centre: What is a Local Internet Registry (LIR)? Accessed: 2016-12-25.
Retrieved from: https://www.ripe.net/manage-ips-and-asns/resource-management/faq/independent-resources/phase-three/what-is-a-local-internet-registry-lir

[17] Réseaux IP Européens Network Coordination Centre: IPv6 Address Allocation and Assignment Policy. October 2015. Accessed: 2016-12-25.
Retrieved from: https://www.ripe.net/publications/docs/ripe-655

[18] de Velde, G. V.; Popoviciu, C.; Chown, T.; et al.: IPv6 Unicast Address Assignment Considerations. Request for Comments: 5375. December 2008. Accessed: 2017-02-25.
Retrieved from: https://tools.ietf.org/html/rfc5375

[19] Wang, M.; Deering, S.; Hain, T.; et al.: Non-random generator for IPv6 tables. In *Proceedings. 12th Annual IEEE Symposium on High Performance Interconnects*. Aug 2004. pp. 35–40. doi:10.1109/CONECT.2004.1375198.

[20] Zheng, K.; Liu, B.: V6Gene: a scalable IPv6 prefix generator for route lookup algorithm benchmark. In *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA'06)*, vol. 1. April 2006. ISSN 1550-445X. pp. 1–6. doi:10.1109/AINA.2006.344.