Chapter 5

Results and Discussion

This chapter explains the results obtained after carrying out the Data Analysis process. It demonstrates key reuse across numerous hosts through the help of graph visualisations and other tables to provide an in-depth analysis of the clusters. This chapter also provides insights into the results after the optimisation techniques were used and how they affected the run time and memory usage of the program.

5.1 Overview of Results

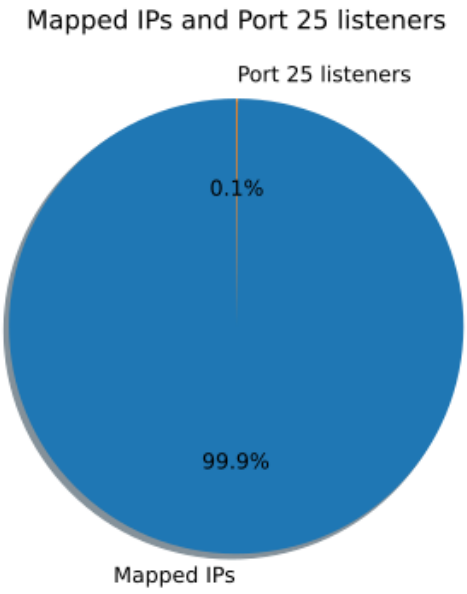

Figure 5.1: IE Port 25 listeners

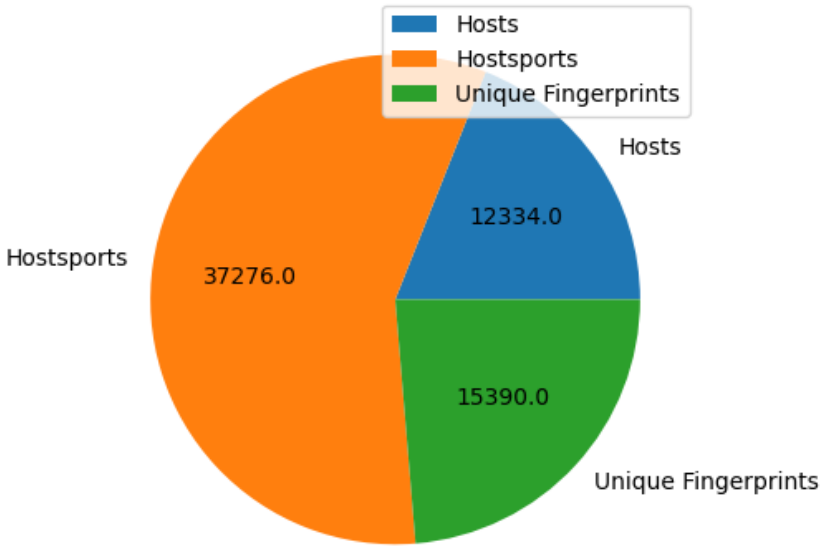

Figure 5.2: Hosts vs Host Ports vs Fingerprints

Figure 5.1 shows the number of IP addresses identified by ZMap as port 25 listeners. Out of the total IPs for Ireland (16421058), only 17,665 were identified as mail servers making it 0.1% of the total IPv4s assigned to Ireland.

12,333 hosts did some cryptography out of the 17,665 hosts identified as port 25 listeners, as indicated by 5.2. Out of 12,333, there were about 37,276 hosts-ports combinations that did some cryptography, but there were only 15,390 unique fingerprints seen throughout the scans. As only 41% were unique, the conclusion that there is key sharing among hosts can be made. The sections below further analyse this key reuse and provide analysis on some of the interesting clusters found.

5.2 Protocol Versions

5.2.1 Cryptography per port Count

5.3 depicts how many ports do some sort of cryptography across all IP addresses. Surprisingly, only 32 hosts on port 587 were found to be offering cryptographic services, and while port 443 had been most as expected as they operate the HTTPS protocol, which is primarily webservers.

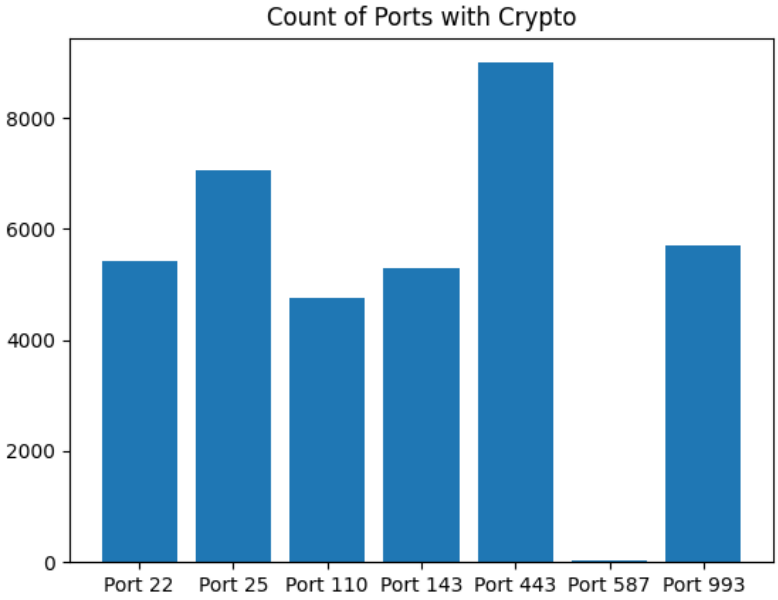

Figure 5.3: Ports that offer Crpto

5.2.2 SSH Versions

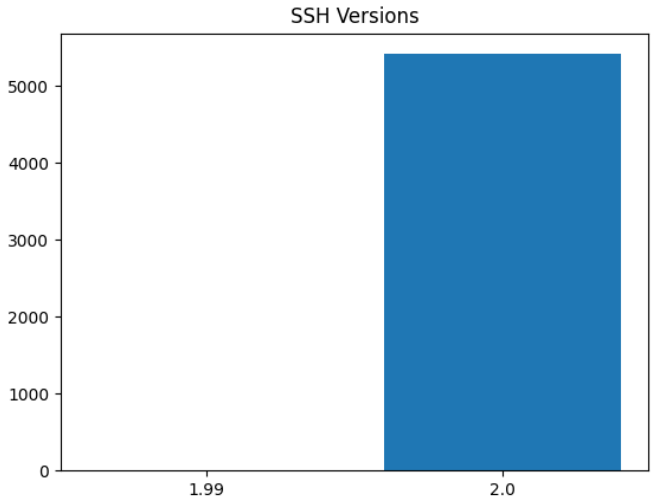

Figure 5.4: SSH Versions

Figure 5.4 above represents the SSH versions seen throughout the scans. Only two SSH versions are operating with only one host using them. SSH 1.9 and the rest of them using SSH 2.0. SSH 2.0 was introduced in 2006 and is the current standard version of SSH introduced by the IETF. It provides significant improvements in terms of efficiency and security. SSH versions 1.0 and 2.0 are incompatible. Hence, SSH 1.9 was introduced by the IETF to provide some backward compatibility between the two versions.

5.2.3 TLS Versions

port	SSLv3	TLSv1.0	TLSv1.1	TLSv1.2	Total
p25	0	120	4	6930	7054
p110	1	166	3	4600	4770
p143	0	179	3	5108	5290
p443	0	266	3	8729	8998
p587	0	2	0	30	32
p993	0	144	7	5563	5714
Total	1	877	20	30960	31858

Table 5.1: TLS Versions

Table 5.1 shows the TLS versions seen throughout the scans irrespective of the IP address belonging to a cluster. Most of the TLS versions seen are TLS 1.2, followed by TLS 1.0, with the least SSLv3. There was no instance of TLS 1.3 in our scans. Surprisingly, it was seen that a significant number of hosts are still operating TLS 1.0, and some are operating TLS 1.1 even though TLS 1.0 and TLS 1.1 have been depreciated by the IETF. The reason why they were depreciated is because of major security flaws associated with the old versions. Some of the reasons behind the depreciation of TLS 1.0 and 1.1 are:

- Old TLS versions require implementation using old cipher suites that are no longer desirable from a security point of view.
- Old versions do not support the modern and recommended cipher suites.
- The integrity of the handshake and the authentication process depend on SHA-1 hashes and signatures, respectively, which can easily be broken and superseded with SHA-2.

TLS 1.0 was released in 1999 and is considered to be the weakest of all TLS versions available, and the IETF does explicitly not permit the use of it [23]. This is because it is known to suffer from attacks like the BEAST due to improper implementation of the Cipher Block Chaining [31].

Even though TLS 1.2 has also been upgraded by TLS 1.3, it was not seen during the scans. This might be because TLS 1.3 upgrade is one of the most significant upgrades as compared to other TLS versions. The new upgrade introduced new features like 0-RTT that provide significant speed upgrades during a TLS connection establishment. Since TLS 1.3 was drastically different, and due to middleboxes all over the internet, the TLS handshake might seem like a TLS 1.2 connection even though it is TLS 1.3 [28].

5.2.4 TLS Cipher Suites

The TLS Cipher Suites seen for the selected clusters:

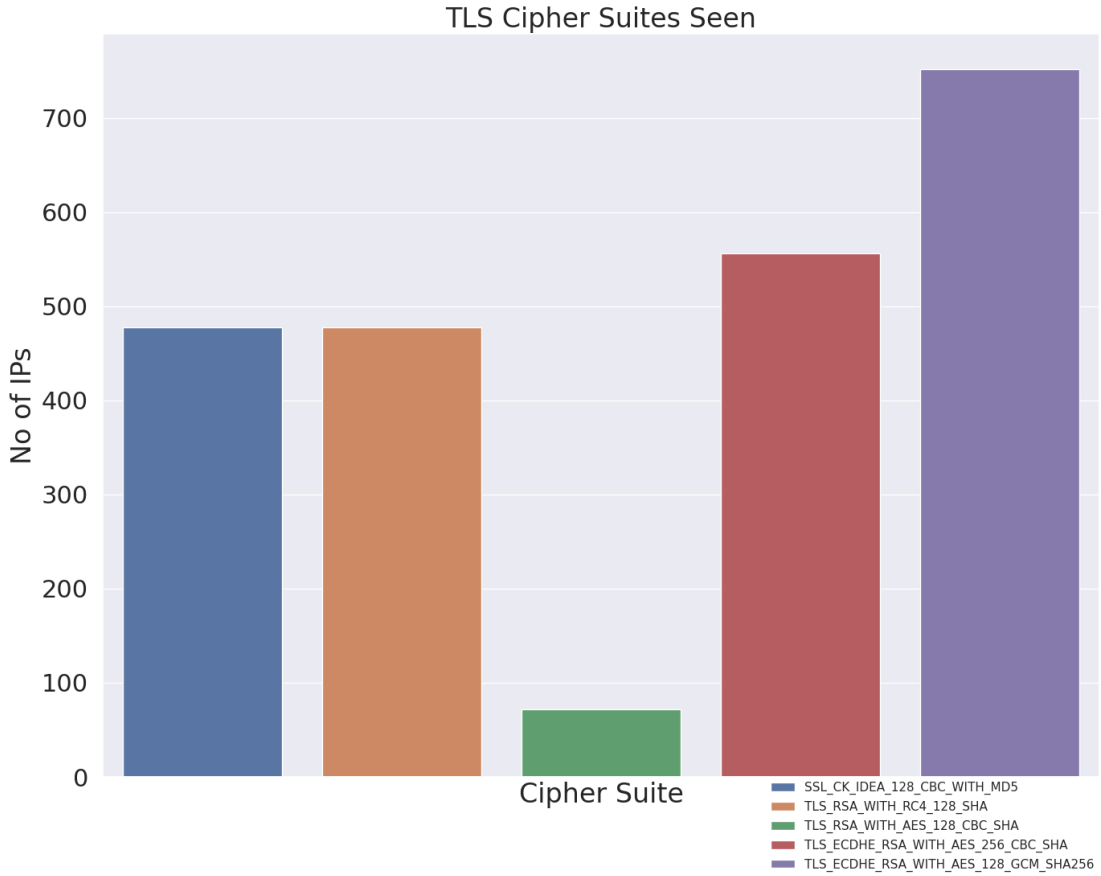

Figure 5.5: TLS Cipher Suites

Table 5.5 shows the TLS cipher suites we see for the selected clusters. The most common cipher suite was found to be `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` followed by `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`. There was a considerable amount of hosts using `TLS_RSA_WITH_RC4_128_SHA` which is known to have cryptographic weaknesses. The IETF prohibits the use of any RC4 based Cipher Suite as they do not provide desired security [25]. The least used Cipher Suite found to be was `TLS_RSA_WITH_AES_128_CBC_SHA`.

5.3 Key Reuse Results

There were a total of 5809 collisions found producing about 1049 clusters in total. The graphs below are a few of the selected clusters from the results. Some of them were selected based on the number of hosts, while others were selected randomly.

5.3.1 Cluster 43

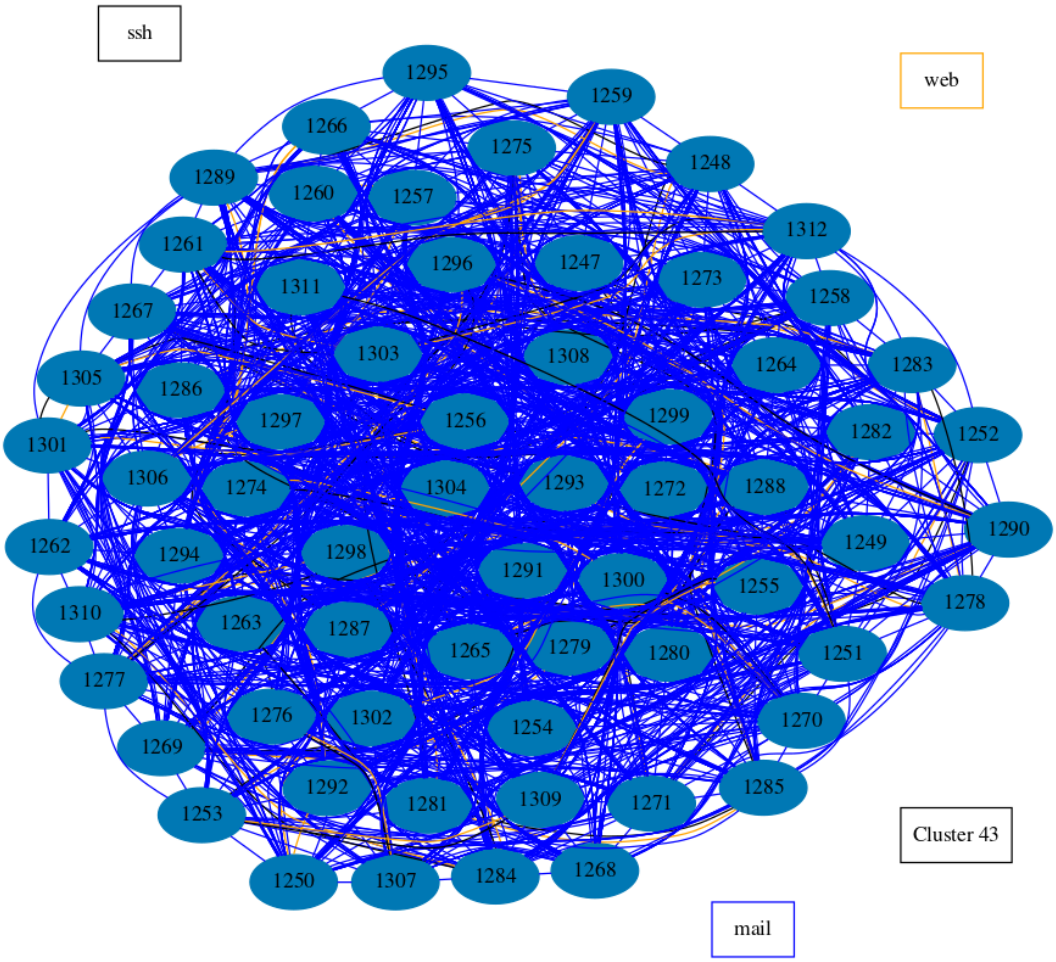

Figure 5.6: Cluster 43

Figure 5.6 represents Cluster 43 from the results. It is a cluster consisting of 66 hosts sharing the same keys for SMTP, SSH and HTTPS protocol. All of the hosts belong to same Autonomous System in this case. There are about 722 host-port combinations with 86 of them being SSH. There were only 14 SSH unique keys seen for the 86 host-port combiantions. Out of the 722 host-port combiantions, 636 were TLS ports with only 22 unique TLS keys seen. The AS in question here is a web hosting service with a local presence.

5.3.2 Cluster 72

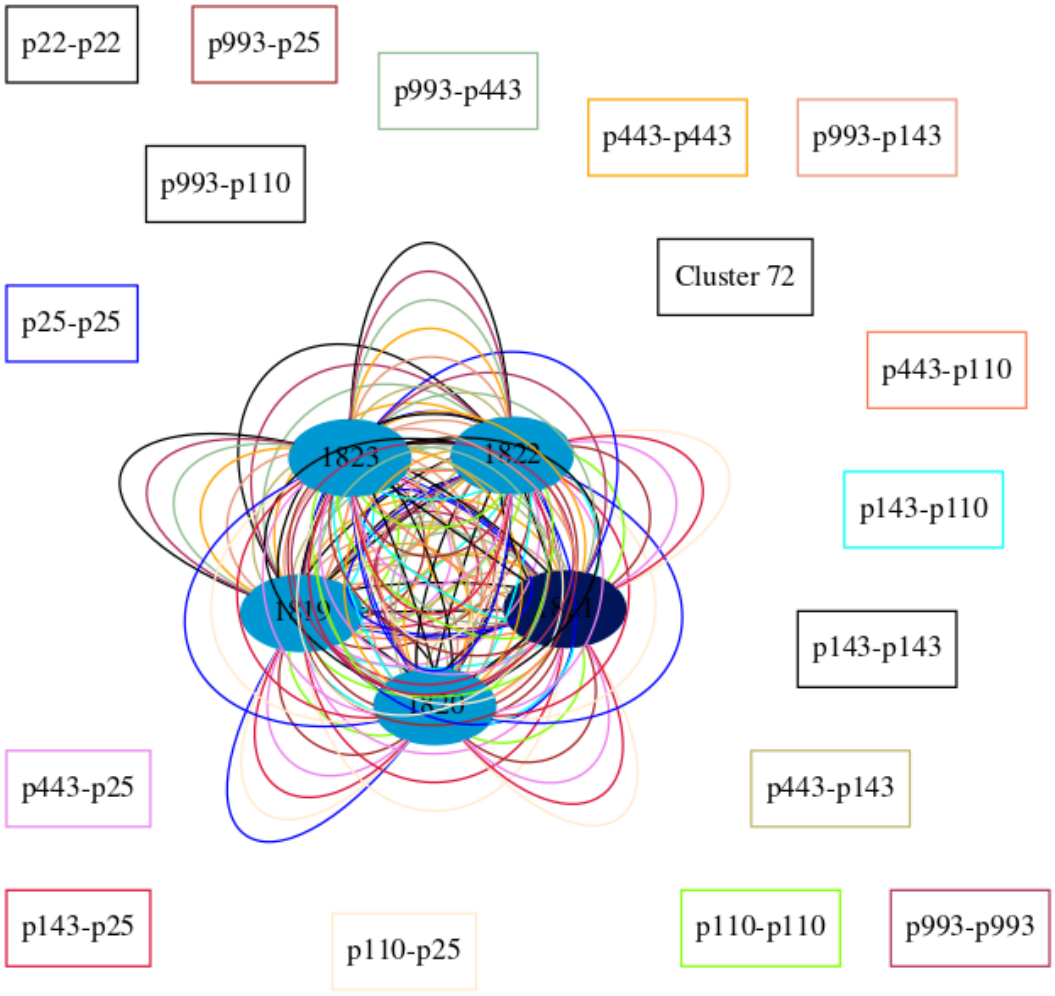

Figure 5.7: Cluster 72

Figure 5.7 is an interesting one. We see key sharing across almost every pair port combination. We see five hosts in this cluster out of which four belong to the same AS. There is key sharing for across all mail protocols but we even see cross protocol key sharing among hosts for HTTPS and SMTP, IMAP and POP3. Upon inspection of the cluster data it was found that the the four ASes that are same is againn a webhosting service while the other AS seems to belong to a Telecommunications company.

5.3.3 Cluster 133

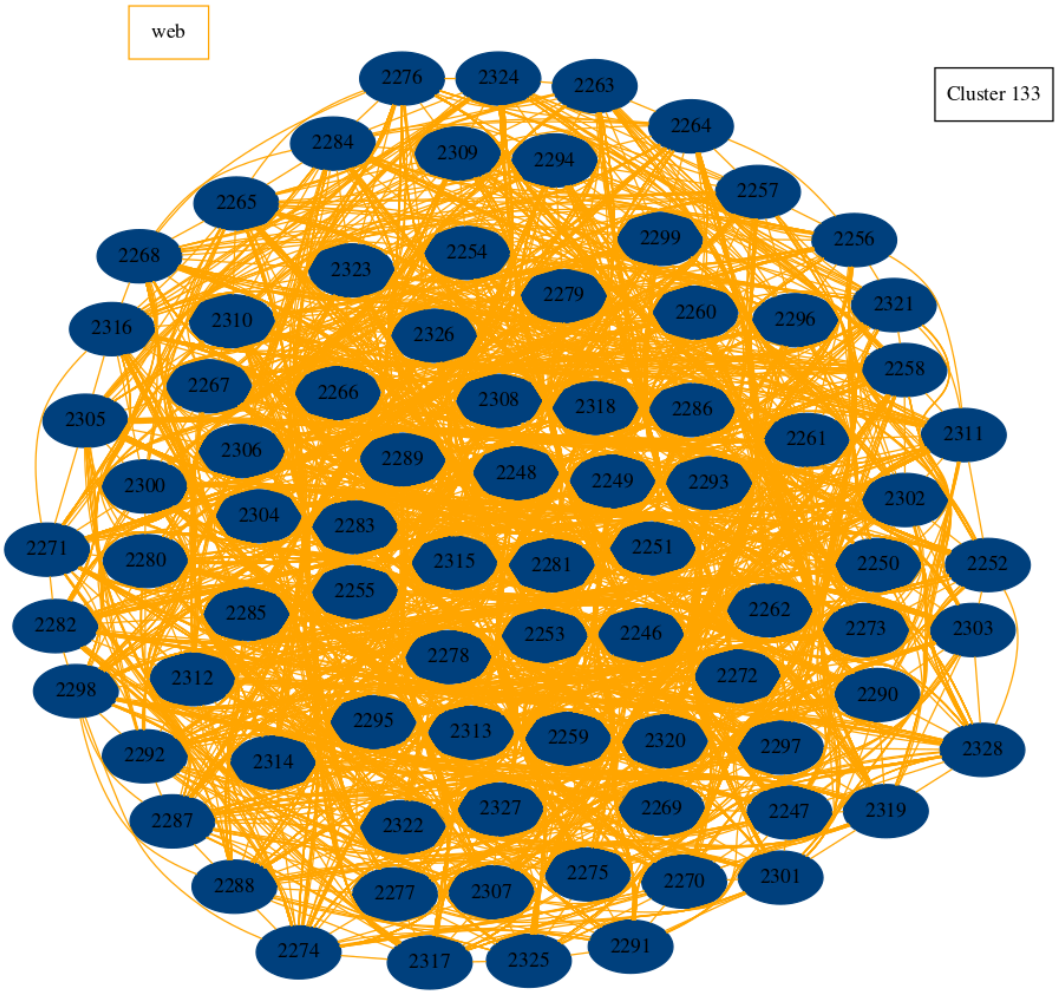

Figure 5.8: Cluster 133

Figure 5.8 represents Cluster 133 from the results and it consists of 83 hosts sharing web server keys (HTTPS). This is one of the busier clusters found in the results and there a few clusters larger than this. All of the 83 hosts belong to the same AS, a global web hosting service and share keys for only port 443. There are about 166 host-port combinations all sharing the same TLS key.

5.3.4 Cluster 148

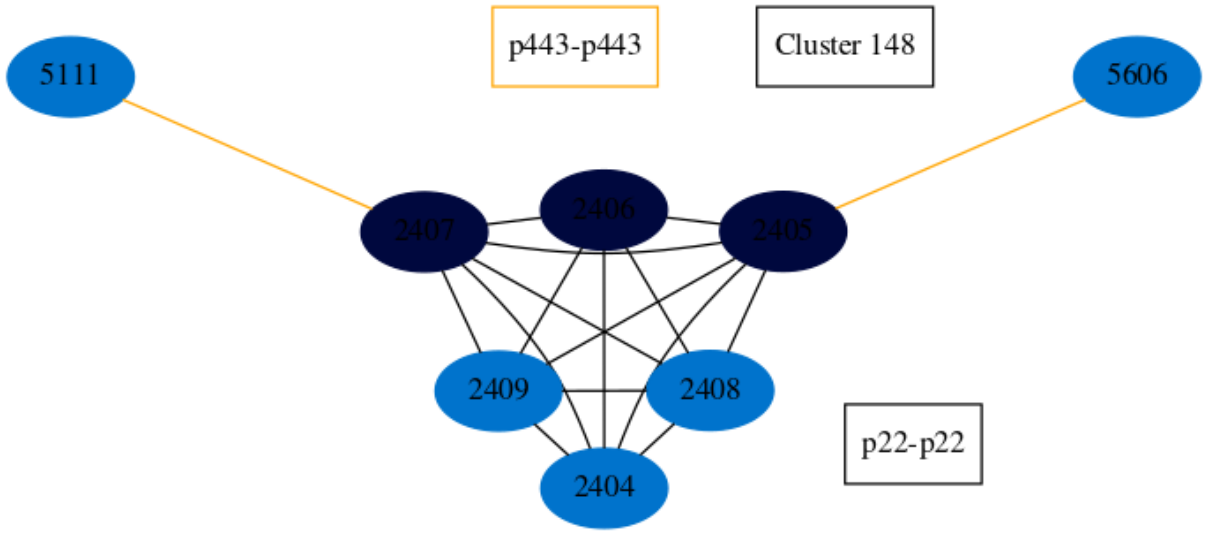

Figure 5.9: Cluster 148

Figure 5.9 represents Cluster 148 and and consists of eight hosts belonging to two differnt ASes. Out of the eight, three belong to the same AS and 5 to a differnt one. All hosts in the middle shares keys for SSH while the the two host 5111 and 5606 at the edges share keys with host 2607 and 2605 for port 433 respectively. There are about 22 host-port combinations seen out of which 12 are classified as SSH and the remaining 10 are TLS. Only 1 SSH key is being shared among the 12 host-port combinations while 3 TLS keys being shared for 10 host-port combiantions. The AS with three hosts belong to local ISP while the AS with 5 hosts belong to a telecommunications entity with a huge local presence.

5.3.5 Cluster 536

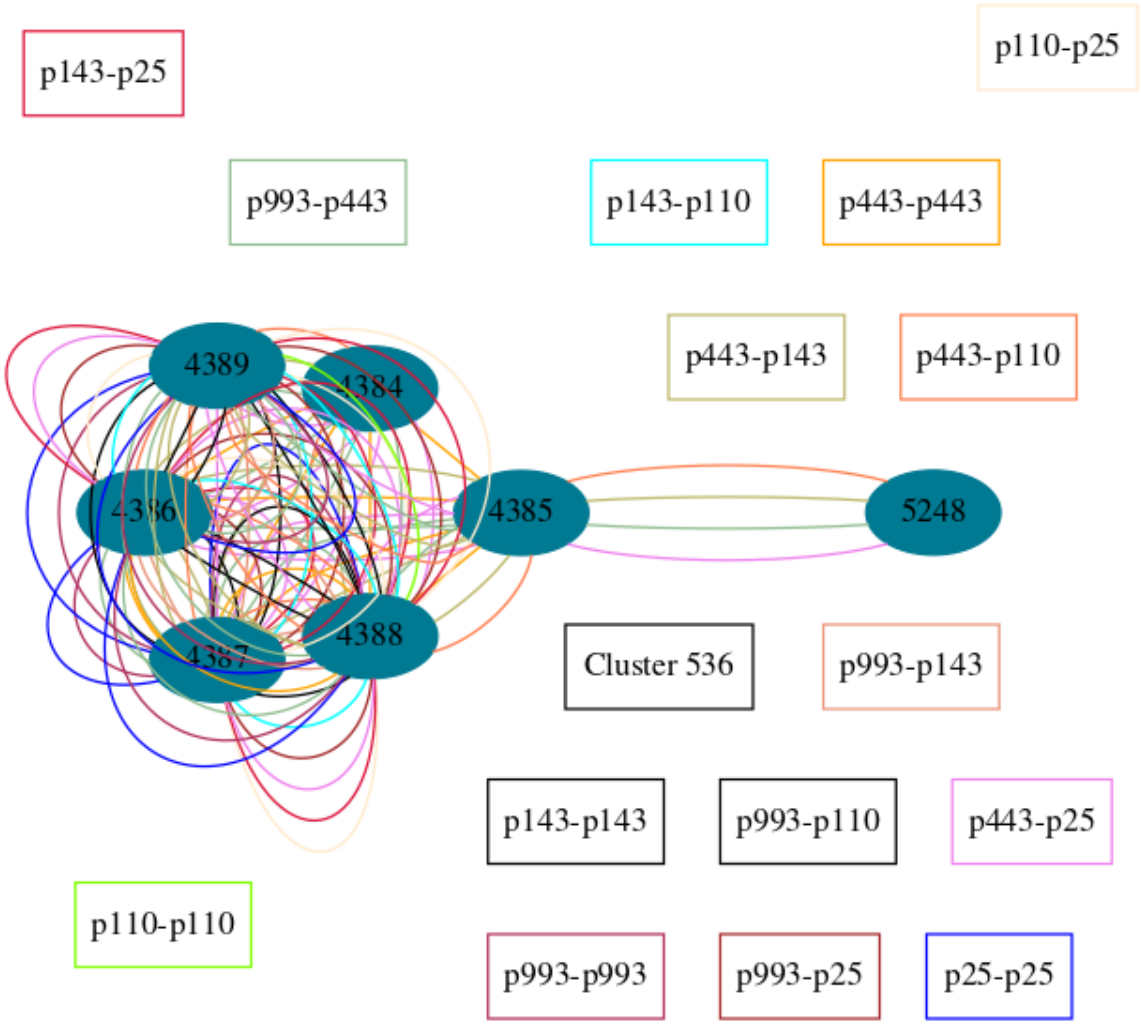

Figure 5.10: Graph 536

Figure 5.10 represents Cluster 536 and has about 7 hosts all belong to the same AS. There are seven hosts with 58 hosts-port combinations and only only four unique TLS keys. The max key usage is 38 times in the cluster.

5.3.6 Cluster 5 and Cluster 15

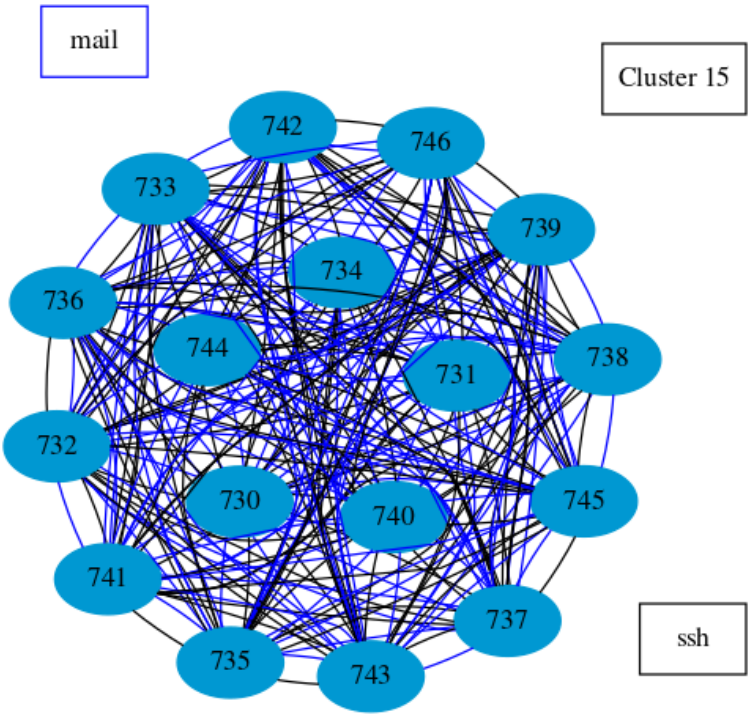

Figure 5.11: Cluster 15

Cluster 5 is the biggest SSH cluster found in our reuslts. It has 226 hosts all belong to the same AS with about 1322 host port combinations. Out of the total host port combiantions, 418 are SSH ports with only one SSH key being shared among all. The rest of the 904 host-ports combiantions are TLS ports and again we see only one TLS key among them.

Figure 5.11 is the biggest pure SSH cluster and has about 15 hosts in total belong to the same AS. There a total of 68 hostport combinations with 34 of them being SSH and the other 34 being TLS ports. For both SSH and TLS we only see one key being reused among all hosts respectively.

5.4 Post Refactoring and Optimisation

The sections discusses the results obtained after code refactoring as mentioned in section 4.7. The profiling is done again after making the changes and modifications in the program, and after using the new DNS setup. There is also a timing graph presented that compares the average time per IP spent analysing using the two DNS setups.

```
rs@rs:~/surveys/rsstuff$ ./memory_profile.py
Tue Apr 12 02:20:04 2022    /home/rs/survey_results/new_setup_profile

         237109343 function calls (187821058 primitive calls) in 3750.811 seconds

   Ordered by: internal time
   List reduced from 2184 to 15 due to restriction <15>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    43952 2135.976    0.049 2137.045    0.049 {built-in method _socket.gethostbyname}
    17647 1111.867    0.063 1112.373    0.063 {built-in method _socket.gethostbyaddr}
        1  244.595  244.595 3750.811 3750.811 SameKeys.py:1(<module>)
6407352/105882   23.982    0.000   50.327    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/decoder.py:141(decode)
  5974621   21.024    0.000   25.256    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/reader.py:190(_read_node)
45503685/16193627   15.464    0.000   18.983    0.000 /usr/lib/python3.10/json/encoder.py:333(_iterencode_dict)
    52448   13.905    0.000   13.905    0.000 {built-in method builtins.print}
    17647    9.857    0.001    9.857    0.001 /usr/lib/python3.10/json/decoder.py:343(raw_decode)
    17647    8.376    0.000   38.150    0.002 /home/rs/surveys/SurveyFuncs.py:691(mm_setup)
    70596    6.681    0.000    6.872    0.000 {built-in method io.open}
467844/105882    6.297    0.000   42.340    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/decoder.py:85(_decode_map)
1394148/462479    5.986    0.000   31.131    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/decoder.py:93(_decode_pointer)
    52941    4.986    0.000   22.162    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/reader.py:39(__init__)
 16193630    4.632    0.000   24.292    0.000 /usr/lib/python3.10/json/encoder.py:413(_iterencode)
   828334    4.329    0.000    6.946    0.000 /usr/lib/python3/dist-packages/dateutil/parser/_parser.py:83(get_token)
```

Figure 5.12: Memory Profile after Refactoring

Figure 5.12 represents the memory profiling after refactoring and optimisation as depicted in the sections above. The runtime for data processing and analysis reduced from 73816 seconds to 3750 seconds. After refactoring, the number of recursive calls was reduced, contributing to optimising the memory consumption as recursive methods can take up a lot of memory. The new DNS setup significantly improved the runtime of the program, and this might be due to the following factors:

- Typical DNS setups rely on servers supplied by ISPs that might be slow.
- Configurations for caching might not be implemented properly by the ISP, which may contribute to slow lookups. [30]

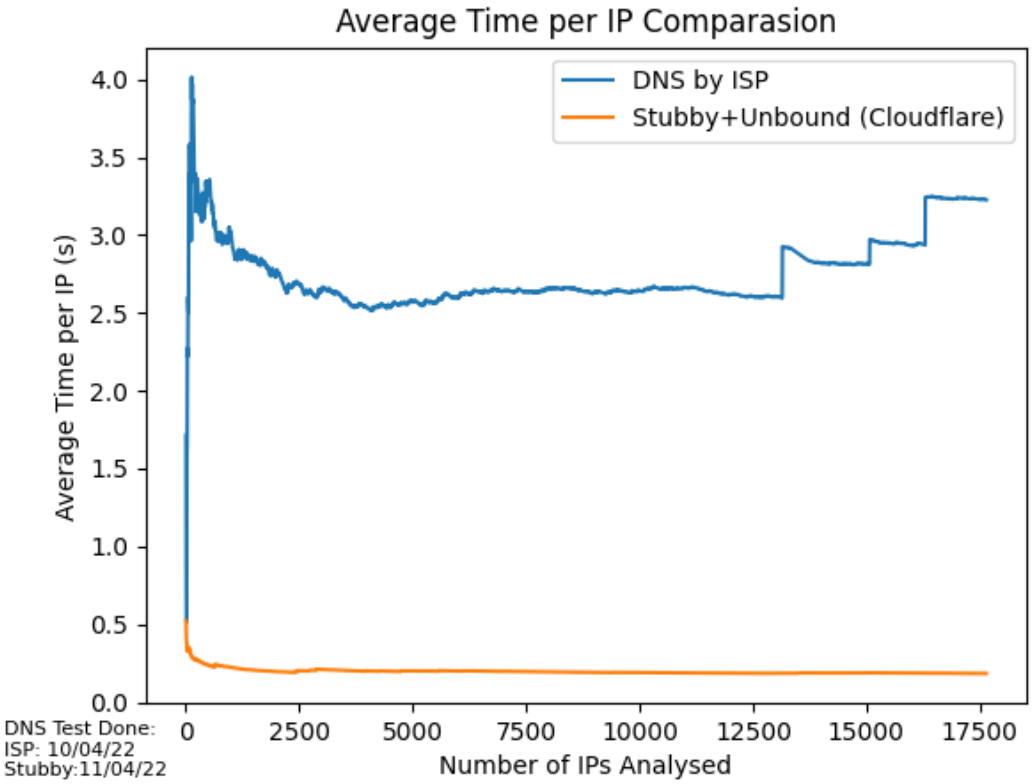

Figure 5.13: DNS Setup Comparisons

Figure 5.13 above shows the difference between the average time spent per IP address when using the two DNS setups. The average time processing each IP was bought down to 0.15s from 3.4s when using the default setup. This might be due to the following reasons:

- Using Unbound, one can get more control over their DNS setup as it allows you to configure the DNS servers you might want to use.
- It will also cache all queries made for faster lookups the next time.
- Using Stubby allows us to use DNS over TLS for sending queries to resolvers. It allows for increased privacy.
- Since Unbound is not as advanced as Stubby and does not allow us to use the same TLS connection for queries, a combination of the two can help speed things up. Stubby can use the same TLS connection to make multiple queries, saving the overhead and time to open up a new connection for each query. [14, 26]