IPv6 Deployment - Monitoring and Visualisation

by Daniel Martinho-Corbishley & James Cotterell & Jack Tench

March 5, 2014
A group design project report submitted for the award of Master of Computer Science

by Daniel Martinho-Corbishley & James Cotterell & Jack Tench

The primary focus of the Group Design Project is to redevelop the current IPv6 Matrix website, to provide ease of access to a comprehensive set of analysed data with the goal of helping provide data for publication and to help present the case for IPv6 adoption to decision makers.

This project has achieved its main aim of redesigning the IPv6Matrix.org website to provide better and more complete visualisations of the data gathered by the IPv6Matrix Crawler. The conversion from raw data to a format usable by the website has been greatly improved, providing a larger amount of processed and summarised data in a similar amount of time. The website, implemented using Node.js, uses maps, graphs and tables to display the significant quantity of data available with an emphasis on historical and geographical representation.

Some aspects of the project discussed in the original brief have not been completed due to the focus of the project being on the completion of the website, combined with the amount of work required to achieve this. There are therefore a number of opportunities to improve and extend the Crawler, Data API and Website in further work.
## Contents

### Acknowledgements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xv</td>
<td></td>
</tr>
</tbody>
</table>

### 1 Introduction

<table>
<thead>
<tr>
<th>1.1</th>
<th>IPv6Matrix.org</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Project aims</td>
</tr>
<tr>
<td>1.3</td>
<td>Project scope</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### 2 Research

<table>
<thead>
<tr>
<th>2.1</th>
<th>Discussion with client</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Open Linked Data</td>
</tr>
<tr>
<td>2.3</td>
<td>Existing implementation</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Crawler</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Data analysis</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Website</td>
</tr>
<tr>
<td>2.4</td>
<td>Other IPv6 data sites</td>
</tr>
<tr>
<td>2.5</td>
<td>Data visualisation examples</td>
</tr>
<tr>
<td>2.6</td>
<td>Website technologies</td>
</tr>
<tr>
<td>2.6.1</td>
<td>Using JavaScript</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Web servers</td>
</tr>
</tbody>
</table>

### 3 Design

<table>
<thead>
<tr>
<th>3.1</th>
<th>Crawler</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Data processing</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Database design</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Analysis</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>CSV conversion</td>
</tr>
<tr>
<td>3.2.2.2</td>
<td>Summary generation</td>
</tr>
<tr>
<td>3.2.2.3</td>
<td>Selecting data points</td>
</tr>
<tr>
<td>3.3</td>
<td>API design</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Routes, resources &amp; queries</td>
</tr>
<tr>
<td>3.3.1.1</td>
<td>IPv6 Hosts</td>
</tr>
<tr>
<td>3.3.1.2</td>
<td>Dual Stack</td>
</tr>
<tr>
<td>3.3.1.3</td>
<td>Ping</td>
</tr>
<tr>
<td>3.3.1.4</td>
<td>Path</td>
</tr>
<tr>
<td>3.3.1.5</td>
<td>Reachability</td>
</tr>
<tr>
<td>3.3.1.6</td>
<td>Domains</td>
</tr>
<tr>
<td>3.3.1.7</td>
<td>Domain Search</td>
</tr>
</tbody>
</table>
CONTENTS

3.3.1.8 Country Overview .............................................. 26
3.3.2 Data formats .................................................. 27
  3.3.2.1 JSON .................................................. 27
  3.3.2.2 XML .................................................. 27
  3.3.2.3 HTML ............................................... 28
  3.3.2.4 CSV ............................................... 28
  3.3.2.5 Generic request ........................................ 29
3.4 Website design ................................................ 29
  3.4.1 Structure & navigation ...................................... 29
  3.4.2 Data visualisation & interaction .......................... 30
    3.4.2.1 Data selection ..................................... 31
  3.4.3 User Interface ................................................ 31

4 Technical Approach .............................................. 35
  4.1 Data processing ................................................ 35
    4.1.1 CSV conversion ........................................ 35
      4.1.1.1 Performance optimisation ............................ 36
    4.1.2 Summary generation ..................................... 37
  4.2 Node.js web server ............................................ 41
    4.2.1 Express middle-ware .................................... 41
    4.2.2 Jade template middle-ware .............................. 43
    4.2.3 Data API implementation ............................... 45
  4.3 JavaScript web-app ............................................. 46
    4.3.1 Frameworks .............................................. 46
    4.3.2 Libraries ............................................... 48
    4.3.3 MV* implementation .................................... 49
      4.3.3.1 Models ........................................... 50
      4.3.3.2 Router ........................................... 50
      4.3.3.3 Views ............................................ 51
    4.3.4 Interface implementation ............................... 53
    4.3.5 Data visualisations .................................... 55
      4.3.5.1 Map .............................................. 55
      4.3.5.2 Graphs ........................................... 57
    4.3.6 Deployment ............................................... 60
  4.4 Deployment & packaging ..................................... 61

5 Testing .............................................................. 63
  5.1 Website memory usage ......................................... 63
  5.2 Data summary .................................................. 65
  5.3 Data API ......................................................... 66
    5.3.1 Performance ............................................. 66

6 Evaluation ........................................................ 67
  6.1 Data processing ................................................ 67
  6.2 API ............................................................ 68
  6.3 Website ........................................................ 69
    6.3.1 Data representation & accessibility .................... 69
List of Figures

2.1 Some of the data available at Vyncke.org/ipv6status ........................................ 12
2.2 IPv6 deployment visualisation at My.ES.net/sites/ipv6 ......................................... 13
2.3 Akamai.com/ipv6 ........................................................................................................ 14
2.4 Bubble Chart ................................................................................................................ 15
2.5 With the use of some colour, a globe could be an interesting way to display information on IPv6 deployment ................................................................. 16
2.6 D3 Show Reel ................................................................................................................ 17
3.1 Wireframe prototypes 1 ................................................................................................. 32
3.2 Wireframe prototypes 2 ................................................................................................. 33
4.1 Custom symbols font from Fontello.com ..................................................................... 54
5.1 Memory Test 1: Web-app memory allocations (as-is) over time ................................. 64
5.2 Memory Test 2: Web-app memory allocations excluding mapView over time .......... 64
5.3 Memory Test 3: Web-app memory allocations excluding nvd3 charts over time .......... 65
8.1 Commits by author over time ....................................................................................... 81
8.2 Commits by day of the week ......................................................................................... 82
8.3 Commits by hour of the day ......................................................................................... 82
B.1 menuView DOM element ............................................................................................. 97
B.2 sideView DOM element ............................................................................................... 97
B.3 controllerView DOM element ..................................................................................... 98
B.4 sliderView DOM element ............................................................................................ 98
B.5 settingsView DOM element ......................................................................................... 98
B.6 searchView DOM element .......................................................................................... 99
B.7 mapView DOM element - Orthographic projection .................................................. 100
B.8 mapView DOM element - Mercator projection ......................................................... 100
B.9 graphView DOM element ........................................................................................... 101
B.10 tableView DOM element ......................................................................................... 101
B.11 Advance options checkbox tree built using jQuery Tree ........................................... 101
C.1 Trello Scrum style task board ..................................................................................... 103
C.2 Advanced settings ........................................................................................................ 104
C.3 Country Page ............................................................................................................... 105
C.4 Domain Page ................................................................................................................ 106
C.5 Dual Stack Page ................................................. 107
C.6 Splash Page ................................................. 108
C.7 IPv6 Hosts .................................................. 109
C.8 IPv6 Hosts Continued ........................................ 110
C.9 Ping Data ................................................. 111
C.10 Reachability ............................................... 112
C.11 Domain Search Typeahead ................................ 113
C.12 Domain Search Data ......................................... 114
## List of Tables

<table>
<thead>
<tr>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Table showing HTTP load testing results for both the Data API and static website files</td>
<td>66</td>
</tr>
<tr>
<td>7.1 Existing Crawler Issues</td>
<td>76</td>
</tr>
<tr>
<td>7.2 Existing API Issues</td>
<td>76</td>
</tr>
<tr>
<td>7.3 Existing Website Issues</td>
<td>76</td>
</tr>
<tr>
<td>7.4 Crawler Extensions</td>
<td>76</td>
</tr>
<tr>
<td>7.5 API Extensions</td>
<td>77</td>
</tr>
<tr>
<td>7.6 Website Extensions</td>
<td>77</td>
</tr>
<tr>
<td>8.1 Per author git commit statistics</td>
<td>82</td>
</tr>
<tr>
<td>F.1 Structure of table domainPenetration_summary</td>
<td>123</td>
</tr>
<tr>
<td>F.2 Sample content of table domainPenetration_summary</td>
<td>123</td>
</tr>
<tr>
<td>F.2 Sample content of table domainPenetration_summary (continued)</td>
<td>124</td>
</tr>
<tr>
<td>F.3 Structure of table geoip_summary</td>
<td>124</td>
</tr>
<tr>
<td>F.4 Sample content of table geoip_summary</td>
<td>124</td>
</tr>
<tr>
<td>F.5 Structure of table IpDuality_summary</td>
<td>125</td>
</tr>
<tr>
<td>F.6 Sample content of table IpDuality_summary</td>
<td>125</td>
</tr>
<tr>
<td>F.7 Structure of table path_summary</td>
<td>125</td>
</tr>
<tr>
<td>F.7 Structure of table path_summary (continued)</td>
<td>126</td>
</tr>
<tr>
<td>F.8 Sample content of table path_summary</td>
<td>126</td>
</tr>
<tr>
<td>F.9 Structure of table ping_summary</td>
<td>126</td>
</tr>
<tr>
<td>F.10 Sample content of table ping_summary</td>
<td>127</td>
</tr>
<tr>
<td>F.11 Structure of MySQL table HostData</td>
<td>127</td>
</tr>
<tr>
<td>F.11 Structure of MySQL table HostData (continued)</td>
<td>128</td>
</tr>
<tr>
<td>F.12 Sample content of MySQL table HostData</td>
<td>128</td>
</tr>
<tr>
<td>F.13 Structure of MySQL table DomainPenetration</td>
<td>128</td>
</tr>
<tr>
<td>F.13 Structure of MySQL table DomainPenetration (continued)</td>
<td>129</td>
</tr>
<tr>
<td>F.14 Sample content of MySQL table DomainPenetration</td>
<td>129</td>
</tr>
<tr>
<td>F.15 Structure of MySQL table Reachability</td>
<td>129</td>
</tr>
<tr>
<td>F.15 Structure of MySQL table Reachability (continued)</td>
<td>130</td>
</tr>
<tr>
<td>F.16 Sample content of MySQL table Reachability</td>
<td>130</td>
</tr>
<tr>
<td>F.17 Structure of MySQL table Domains</td>
<td>130</td>
</tr>
<tr>
<td>F.17 Structure of MySQL table Domains (continued)</td>
<td>131</td>
</tr>
<tr>
<td>F.18 Sample content of MySQL table Domains</td>
<td>131</td>
</tr>
</tbody>
</table>
Listings

4.1 Host Type Summary SQL .......................................................... 38
4.2 Ping Summary SQL ................................................................. 39
4.3 Path Summary SQL ................................................................. 39
4.4 Reachability Summary SQL ...................................................... 40
4.5 Domain Summary SQL ............................................................ 40
4.6 Route map example for Data API with Express ......................... 42
4.7 Example Jade template web page .............................................. 44
4.8 data API XML format Jade template ........................................ 45
4.9 Example of building a SQL query in javascript using node-sql .... 45
4.10 Capturing links for the History API ......................................... 51
4.11 mainView setViews() function ................................................ 51
4.12 Example text template script in HTML .................................... 54
4.13 Example underscore template string in JavaScript .................... 55
4.14 Using topoJSON to read the world map data into d3 ............... 55
4.15 Calculating the SVG scale for a selected country ...................... 56
4.16 Calculating colours for each country element based on mapModel data ........................................ 56
4.17 Setting variables to use in data processing based on the page being displayed ........................................ 57
4.18 The parent initialize function is called before setting the chart style axis formatting and labelling ........................................ 58
4.19 Extra jQuery templating and utility functions required for the buttons in ButtonGraphView ........................................ 58
4.20 Resize function used to correctly adjust graph sizes when web page is scaled ........................................ 59
4.21 RequireJS script tag ............................................................... 60
4.22 Example RequireJS configuration file ....................................... 60
4.23 Web-app build.js configuration file .......................................... 61
A.1 Deployment instructions ......................................................... 94
E.1 SQL to build new MySQL ipv6matrix database structure ............. 119
E.2 Sample of Data API URLs for HTTP load testing ....................... 121
Acknowledgements

Thanks to Olivier Crepin-Leblond and Dr. Tim Chown.
Chapter 1

Introduction

IPv6 is the latest specification of the Internet Protocol (IP) and defines an improved addressing scheme for devices to communicate with each other across the internet. It is intended to replace the current IPv4 implementation in order to accommodate the sudden rise in the number of devices connecting to the internet.

For IPv6 to be effective, it must be implemented across all machines involved in accessing and routing internet traffic. Therefore, countries in which the Government has provisioned the modernisation to an IPv6 network infrastructure, have shown a marked increase in IPv6 content roll-out.

Recent initiatives by the Internet Society (ISOC), in particular the World IPv6 Launch Event of June 2012\[32\], have led to a slow but steady growth in IPv6 traffic and IPv6-enabled resources on the Internet. This project continues an existing programme initiated by ISOC, to monitor and present data on the adoption of IPv6 around the globe. The data has so far been shared during conferences, papers and quotes on many occasions. However, it must be made more openly accessible via the internet, to further persuade and educate key decision makers to implement IPv6. Recent initiatives by the Internet Society (ISOC), in particular the World IPv6 Launch Event of June 2012, have led to a slow but steady growth in IPv6 traffic and IPv6-enabled resources on the Internet. This project continues an existing programme initiated by ISOC, to monitor and present data on the adoption of IPv6 around the globe. The data has so far been shared during conferences, papers and quotes on many occasions. However, it must be made more openly accessible via the internet, to further persuade and educate key decision makers to implement IPv6.
1.1 IPv6Matrix.org

The IPv6 Matrix is a suite of tools to measure IPv6 deployment across the internet. It consists of two servers; a back-end ‘IPv6 Crawler’ and a front-end website. The project client, Olivier Crepin-Leblond (chair, ISOC England) has overseen the development of the tools, initially built by a team at Nile University, Egypt.

The IPv6 Crawler runs through the internet’s 1 million most popular domain DNS entries, provided by Alexa.com. The records are crawled at pre-set intervals in order to detect IPv6 enabled Web, Mail, Time and Name servers. All raw data is stored as plain text CSV files, providing ongoing snapshots on the spread of IPv6 accessible content worldwide.

ISOC strives to provide reliable information about the Internet and promote its evolution. The existing website prototyped the display of this collection of data, but the representations produced were rather elementary - “The current Web site falls short of the usability and ergonomics to do justice to the wealth of information which has been collected thus far” - Olivier Crepin-Leblond.

1.2 Project aims

This project aims to catalyse the rate of IPv6 adoption by providing a comprehensive set of analysed data, made openly accessible in a variety of formats, and to develop a website with which to visualise the full range of available data. The project brief specifies an outline of the clients intention for the project, Chapter ???. In order to realise these intentions, the project client has identified three main project goals;

Website  The primary focus of the Group Design Project is to redevelop the current IPv6 Matrix website, providing ease of access to a comprehensive set of analysed data. The main audience consists of the technical community, with the goal of helping provide data for publication and to help present the case for IPv6 adoption to decision makers. It should also be accessible to a wider audience, for educational purposes.

Results must be represented and interpreted on automatically generated Web pages through a series of graphical visualisations, highlighting trends in IPv6 adoption with relation to geographical location and time. Whilst maintaining ease of use, the visualisations should allow for a high level of flexibility and detail, providing the user with a range of parameters with which to filter the data e.g. by geo-located country, top level domain (TLD), service type etc. The exact set of visualisations and parameters will be researched based on client feedback, and will include a variety of interactive maps, graphs and charts.
Open Linked Data API  The next goal is to enable first and third party access to the analysed data, by implementing an API (Application Programming Interface) to provide the information in a multitude of formats. This entails the conversion from the raw data output, gathered by the IPv6 crawler, into useful statistical data. It would also be desirable for the Matrix data to be utilised as an Open Linked Data source, for developers wishing to produce other applications based on the Matrix data, or to combine it with other sources of Open Data.

Packaging  A secondary task is to package the IPv6 Matrix crawler and website into a distributable format. The final package will allow third parties to run the IPv6 Crawler from their own location and provide access to the data via the API. The idea being, that each remote API can then be used as an alternative data source for visualisations on the website. These will provide a better understanding of IPv6 connectivity around the globe.

1.3 Project scope

With such an open ended project, it is important to define the scope under which research and development will be carried out. We outline what is involved for each goal and also discuss what is outside the project scope.

Exisitng operation  The existing crawler and its operation will be investigated to some degree, in order to understand the basics of its functionality and the data it outputs. Furthermore, the previous development team created a bug backlog which will need be reviewed so that any major issues can be fixed before distribution. Current operation of the crawler requires a high level of manual intervention, automating this further should also be investigated. However, far reaching changes to the existing code are out of scope, with any large issues and modifications to be noted for future reference.

Website & API  Both the website and API will require extensive research and implementation, as core aims of the project. The existing website will be considered for the features and data representations it offers, but will not be used for the implementation of a new approach. Similarly, the API will focus on providing Open Linked Data in order to meet the project criteria, leading to the development of a new solution. This also includes the storage and delivery of data, analysed from the raw output of the crawler.

Packaging  All software aspects of the project should be included in the packaging solution, to allow third parties to easily produce their own views of IPv6 connectivity.
The project will not involve the deployment of alternative crawlers or website mirrors. Therefore, the website will initially display local data from IPv6Matrix.org, though it should have the ability to show data from multiple sources.
Chapter 2

Research

2.1 Discussion with client

Two meetings with the client took place to discuss various aspects of the project and better define the scope surrounding the specification.

8/10/13 An initial Skype conversion took place which mainly involved granting the group access to the two project servers and an run through of the existing codebase. The project specification was also discussed in detail and important points were highlighted by Olivier. These include; the ability to view data on a per domain basis and having a similar (and possibly improved) data filtering functionality as the existing website.

An overview of the group’s plan for the project was presented to the client, outlining the basic approach required for each specification point. Project priorities and scope were also discussed, with lower priorities involving finding and possibly fixing bugs with the existing crawler code and data output. Finally, the target audience for the website was clarified to be aimed at the technical community but should also be accessible to the general public.

28/10/13 The second meeting took place face-to-face, post the Progress Seminar 1 presentation, where the group demonstrated a bare-bone prototype of the website and API to generate client feedback. This meeting focused more on the practical aspects of the project such as the implementation and output of the new raw data analysis and generating ideas for various ways to interactively visualise the data. During this meeting the project scope was modified in-line with the current work that had taken place; amending the crawler was agreed to be left out of scope and the packaging of the crawler was set to be a lower priority than initially planned.
The second meeting took place face-to-face, post the Progress Seminar 1 presentation, where the group demonstrated a bare-bone prototype of the website and API to generate client feedback. This meeting focused more on the practical aspects of the project such as the implementation and output of the new raw data analysis and generating ideas for various ways to interactively visualise the data. During this meeting the project scope was modified in-line with the current work that had taken place; amending the crawler bugs and the packaging it were agreed to be a lower priority than initially planned.

2.2 Open Linked Data

As was suggested in the project brief, the implementation of an Open Linked Data API was researched. This began with investigation of existing software designed to provide this service. In particular, the APIs was required to provide mappings between data stored in a relational database and generate Resource Description Framework (RDF) files which describe that data. RDF files available from Open Linked Data APIs generally contain contextual data which enable the files to be 'Machine-readable', where the use of official definitions gives the context for data from different sources to be combined, compared or extended. An example of this would be providing an official definition of a country with geo-located IPv6 data, which could be combined with information about that country, such as population or Gross Domestic Product, allowing further relationships to be established within the data.

While the inclusion of this type of data is useful and can provide interesting combinations of otherwise unrelated data, implementation of an Open Linked Data API would not be suitable for the bulk statistical data being represented on the IPv6Matrix.org website. RDF files include this extra data by defining every entry as a tuple, adding a name and definition for every piece of data, which would dramatically increase the size of the data files being accessed via the API and would be undesirable when creating a responsive website that relies on this data. The research on this area of the project also included to a meeting with Chris Gutteridge, a systems administrator in the ECS Systems Group who is experienced with Open Linked Data APIs. Since there is no existing reference for the type of data being provided by the API required for website operation, one would need to be created alongside the API. Taking this into consideration, Chris recommended that the API be created specifically to meet the needs of the website to ensure it would actually be used. For these reasons the decision was made to make the implementation of an Open Linked Data API outputting RDF a lower priority, instead outputting data in more typical formats such as JSON and XML to reduce the data overhead of normal use of the IPv6Matrix.org website.
In 2006 Sir Tim Berners-Lee defined four rules for the Semantic Web, designed to improve how Open Linked Data was published on the web by guaranteeing readability and understanding of the data. This was later adapted to a 5-star rating system ranging from data made available in any format on the web (1-star), to RDF data complete with data-type definitions and contextual information (5-star). As the API is designed in particular for use with the IPv6Matrix.org website it only meets the criteria to obtain a 3-star rating, which requires that data is provided in a machine-readable, non-proprietary format. A 4-star rating for this API could be achieved with a small amount of extra work, this would require that the data include definitions for every type of data available via the API - through the use of URIs to denote each data-type. There is also the possibility of increasing again to a 5-star rating, however this would require a much greater undertaking with a couple of options for the actual implementation that would achieve this. First, the data available through the API could be put inside an RDF ‘wrapper’, whereby an RDF file containing the contextual information about the datasets being returned alongside a link to the actual dataset itself would be generated. Another method would essentially require the creation of a second API which is capable of exposing all the data collected by the crawler as individual data points with appropriate contextual information. The API used by the IPv6Matrix.org website could then be modified to process and cache summaries of that data to avoid repeated mass requests to the RDF API.

2.3 Existing implementation

2.3.1 Crawler

To gain a decent understanding of the requirements and priorities of the project, significant research was done on the operation of the Crawler in its current implementation, along with the outstanding bug list provided by Olivier. The first task in this research was to understand the data output and how it was generated.

General Operation The list of domains used by the Crawler to generate data is taken from Alexa.com’s list of top million domains. Although the list is updated daily, the Crawler still uses the copy downloaded in 2010 which means that many new or recently popular domains are not present and an increasing number of domains that are no longer active are still included. The domains being used are split into four batches based on Top Level Domain, this ensures that more accurate timestamps can be given to the results and also issues where a crash would make it necessary to restart the crawl from scratch, losing potentially months of runtime, can be avoided. The results of the crawls are output in CSV log files in a folder hierarchy representing the test being run,
Chapter 2 Research

the date and TLD (tld/date/test.log), these are later imported into SQLite database files stored alongside the log files to be used for processing and summary generation.

Data gathered by the crawler involves two broad stages, the first is querying Domain Name Server (DNS) records to find IPv4 and IPv6 addresses. This is followed by the second, where some further testing is performed on the addresses found in the first stage. The DNS lookups carried out in the first stage identify addresses for Web (WWW), Mail (MX), Name (NS) and Time (NTP) servers. Web and Time servers are found via the A and AAAA records (IPv4 and IPv6 respectively) that correspond to certain sub-domains of the domain being crawled. The sub-domains searched for Web servers are: www, www6, six, ipv6 and www.ipv6, while the crawler looks at the ntp and time sub-domains for Time servers. Mail and Name servers have their own entries in DNS records, MX and NS respectively.

Further Processing  The second stage of the crawl then takes the IPv4 and IPv6 addresses to perform further processing. All hosts with IPv4 addresses are geo-located using the GeoIP database available from MaxMind. As this database only contains geo-location information for IPv4 addresses there are no IPv6 only hosts geo-location data, this issue could be resolved in the future as MaxMind have recently released an IPv6 GeoIP database. Similarly to the out of date domain list, the MaxMind geo-location database has also not been updated since 2010. This can result in some incorrect geo-location, evidenced by some hosts being geo-located to the African Regional Industrial Property Organisation which, as of ISO 3166-1 alpha 2, is no longer in use.

The specification for IPv6 reserves certain address prefixes to be used for particular purposes, for example the prefix 2002 denotes a 6to4 address used to send IPv6 packets over an IPv4 network. A list of these prefixes is kept by the Crawler and can be used to categorise any IPv6 address. However, this is not currently implemented correctly and so the data cannot reliably be used.

Comparisons can be made between the routing of IPv6 and IPv4 traffic by pinging dual stack hosts to compare latency and number of hops when using each type. Every dual stack host is tested 5 times with a 2 second time-out with the intention of producing consistent data while keeping the time required to a minimum. This data is displayed as a summary including data on the number of hosts where IPv6 has a lower latency and the average ratio of total IPv6 latency to IPv4 latency, along with the number of hosts requiring fewer hops to reach and the average ratio of IPv6 hops to IPv4 hops.

One potential issue with the method used to identify Web and Time servers is that searching specific sub-domains may not always return accurate data as ‘Wildcard’ or ‘Star’ entries can be included in DNS records that will match on any undefined sub-domain. This means that if a domain has only one web host and a Star record pointing to that same host the Crawler will find seven or eight copies of the same host, potentially
skewing some of the data. This issue could be solved by either only storing unique IPs or by using reverse DNS lookups to confirm the IPs being returned belong to unique domains.

While reverse DNS lookup is performed by the Crawler, this data is not utilised for any purpose. More information about the DNS records is also retrieved by examining Start of Authority (SoA) information, this defines an authority and other properties for a DNS ‘Zone’ which is simply a section of the domain name space. These records contain administrative information determining largely how quickly changes propagate through the domain name system along with a contact for the person responsible for the domain. However, the SoA information is also not used anywhere in the analysed data.

Reachability tests are performed on all Web and Mail hosts by testing TCP connections on ports 25 (SMTP), 80 (HTTP) and 443 (HTTPS). If the server responds to one of these ports it is deemed to be reachable via the associated protocol. The results of reachability testing for these protocols using IPv6 and IPv4 can be compared, showing how the reliability of IPv6 or dual stack hosts has changed in relation to IPv4 hosts. Some of this data may be biased however, as there is no way to determine prior to testing if a host is intended to support the HTTPS protocol. While it can be safely assumed that a domain with an MX and an A or AAAA DNS record should respond on port 25 for Mail and port 80 for Web access. This can make it appear that HTTPS is generally less reliable across all hosts because the response will be the same regardless of whether that protocol is unsupported or unreachable.

The final test implemented on the Crawler is an examination of Transport Layer Security (TLS). This test determines if the TLS cryptographic protocol is implemented by a host and is largely an extension of the reachability test for HTTPS. The TLS test is currently disabled as investigation of security protocols on a large scale can appear malicious and continued probing in this manner is likely to end with the Crawler being blacklisted by network providers which could negatively impact the operation of the Crawler.

**Existing & New Issues** Alongside the investigation into the general operation of the Crawler, a short list of existing issues was provided by Olivier rated from Low to Severe priority. This included issues like the IPv6 prefix type being stored incorrectly and a request to check for email servers on A or AAAA records if there is no entry for MX. Other issues and potential improvements were noted during the course of the project such as having no capability to recover from crashes, if a crash occurred no data about the Crawler’s current state (which test it was running at the time) was stored other than the stack trace output at the end of the log file. This required manual intervention to identify and fix the cause of the crash and restart the last crawl from the start. The Crawler also lacks any automation, the input set of Top Level Domains must be copied from a particular batch into the input file for the Crawler to use. Particular limits on
Chapter 2 Research

the number of concurrent processes must also be set for certain batches as, for large input sets, continuous blocking and waiting issues can occur when the Crawler attempts to perform some of the tests. This could be a major issue as the Crawler should ideally be entirely automated to make best use of the hardware and time available as well as automatic crash recovery or at least notification by email or other means.

There also several opportunities to optimise the operation of the Crawler where data or tests are needlessly duplicated many thousands of times. In particular these opportunities arise where multiple domains are resolved to the same host, for instance where Name Server entries in DNS records make use of popular name server providers such as Dreamhost or Mail Server entries use popular providers like Google. This has resulted in some cases with 600,000 or more entries for ns1.dreamhost.com, which causes 600,000 ping tests, path tests and GeoIP lookups to be run, where one would have been sufficient. This issue could be resolved by maintaining a list of unique hosts or IPs found during the crawl and performing the necessary processing only once, although some consideration would need to made as to whether this gives an incorrect representation of the data and if ns1.dreamhost.com should account for just one host or for 600,000. Regardless of this decision there would still be the ability to make significant optimisations.

2.3.2 Data analysis

In the existing solution, data collection from crawler log files and subsequent analysis of this data is handled by a single python script simpleLog2db.py, which is a cut down version of log2db.py, not currently used due to large amounts of incomplete and non-functional code. It appears that the Nile University students perviously working on the IPv6 Matrix project originally intended to generate static HTML report files, however, this functionality was never completed.

In summary, the simpleLog2db.py script generates a database representation of the crawlers output for each TLDs latest crawl, using these databases it then generates a basic summary database for the latest crawl, overwriting the existing summary database. All databases use the SQLite database engine, which is “the most widely deployed SQL database engine in the world” [REF]. The key advantage of SQLite is in its simplicity - there is no need for a database server process with users, permissions and communication protocols as the database is entirely self contained in a single file on disk. Permissions and write locking are simply managed by the OS file system - if a user has write access to a SQLite file, they can write to the database.

2.3.3 Website

The existing website implementation consists of an Apache web server and a Content Management System (CMS) using Drupal to deliver the static web pages. For the data
representations “a minimalist Python Web Framework” called CherryPy is used,11 in the webserver.py script, to generate an iframe element with some basic data output for the most recently crawled date, this runs independently to the main web server.

The basic website UI and display elements are rendered in HTML using the Drupal CMS engine.[9] This works adequately for delivering the static pages and content, but alone, proves too limited for the display of the dynamic data. The team at Nile University decided to implement the generation of the data representations using the CherryPy framework, in order to provide more advanced functionality such as “interactively filtering the data by TLD and type (NS, MX, etc..)”. A map for displaying geo-located IPv6 adoption around the globe is generated using the Google Charts API,[13] while the remaining charts are rendered as images by webserver.py using the Python Google Chart library.[10] Most pages also feature a very basic table view of the data, in some cases this is only a single row summary. These are used across a set of 5 pages that each display a different metric for the latest set of crawled data. Additionally, the raw data archives can be accessed through the server using the jqGrid JavaScript plugin for jQuery.[34]

The site does not feature a standalone API, making the data difficult to access for third parties. However, a data archives page can access the raw data through jqGrid and a PHP querying script.

Some issues with the existing approach have been noticed during the research investigation. Most notably, a terse documentation for CherryPy and the webserver.py. Inspecting the code files shows a high degree of coupling between underlying server functionality (e.g. generating database query strings) and the generated page output (e.g. HTML document structure). Without corresponding documentation, it was found to be very hard to make sense of this portion of the codebase.

Though the prototyped website does include basic data filtering functionality, it still lacks the functionality and ergonomics required to meet the project’s goals. It was also found that accessing the website would often be slow to deliver pages and sometimes not work at all if the web server had crashed. These problems were found to be difficult to diagnose as the webserver.py codebase is already hard to disambiguate. Therefore, it was concluded that a fresh approach should be taken towards the website, allowing a more comprehensive, accessible and reliable implementation to be developed.

2.4 Other IPv6 data sites

A significant number of websites exist that display similar data to that of IPv6Matrix.org. As many of these websites as possible were investigated to research the types and detail of existing IPv6 deployment data along with how this data was displayed. Many of
the sites visited had interesting and useful visualisations of IPv6 deployment data but lacked in data quantity, either in terms of the time period covered or how detailed the data was. Similarly there were also several with large and detailed datasets, but lacked informative or varied representations of the data. This showed an obvious need for a website with a significant amount of data collected over a long period of time that also had interesting and detailed visualisations.

Figure 2.1 shows the data available at vyncke.org. There is a large amount of data shown, but with only a small number of static graphs that do not give a lot of detail about the data. This dataset is also limited to a sample size of 50 domains and a limited number of tests on DNS records where a status of GREEN, ORANGE or RED is determined for Web, Email and DNS servers for each domain. The user is also able to click on each country to see more data about the domains in that country that were tested, but the data is always grouped by Country and the Service Type. The different colour statuses also include some testing of reachability, where the better ratings require the server to respond to HTTP, SMTP or DNS requests.

<table>
<thead>
<tr>
<th>Web Country</th>
<th>Sample Size</th>
<th>Green</th>
<th>Orange</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>12</td>
<td>11%</td>
<td>9%</td>
</tr>
<tr>
<td>Canada</td>
<td>50</td>
<td>12%</td>
<td>14%</td>
</tr>
<tr>
<td>Australia</td>
<td>30</td>
<td>14%</td>
<td>16%</td>
</tr>
<tr>
<td>United States of America</td>
<td>30</td>
<td>12%</td>
<td>14%</td>
</tr>
<tr>
<td>The Netherlands</td>
<td>20</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>Singapore</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Vietnam</td>
<td>20</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>New Zealand</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Germany</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>India</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>South Africa</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Switzerland</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>France</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Mexico</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Chile</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Russia</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Email Country</th>
<th>Sample Size</th>
<th>Green</th>
<th>Orange</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>12</td>
<td>11%</td>
<td>9%</td>
</tr>
<tr>
<td>Canada</td>
<td>50</td>
<td>12%</td>
<td>14%</td>
</tr>
<tr>
<td>Australia</td>
<td>30</td>
<td>14%</td>
<td>16%</td>
</tr>
<tr>
<td>United States of America</td>
<td>30</td>
<td>12%</td>
<td>14%</td>
</tr>
<tr>
<td>The Netherlands</td>
<td>20</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>Singapore</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Vietnam</td>
<td>20</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>New Zealand</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Germany</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>India</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>South Africa</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Switzerland</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>France</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Mexico</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Chile</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Russia</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DNS Country</th>
<th>Sample Size</th>
<th>Green</th>
<th>Orange</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>12</td>
<td>11%</td>
<td>9%</td>
</tr>
<tr>
<td>Canada</td>
<td>50</td>
<td>12%</td>
<td>14%</td>
</tr>
<tr>
<td>Australia</td>
<td>30</td>
<td>14%</td>
<td>16%</td>
</tr>
<tr>
<td>United States of America</td>
<td>30</td>
<td>12%</td>
<td>14%</td>
</tr>
<tr>
<td>The Netherlands</td>
<td>20</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>Singapore</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Vietnam</td>
<td>20</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>New Zealand</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Germany</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>India</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>South Africa</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Switzerland</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>France</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Mexico</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Chile</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>Russia</td>
<td>10</td>
<td>10%</td>
<td>10%</td>
</tr>
</tbody>
</table>

Figure 2.1: Some of the data available at Vyncke.org/ipv6status

The U.S. Department of Energy website shown in Figure 2.2 demonstrates a simple visualisation of IPv6 deployment data. With a clear legend indicating what the colours in the table mean, the data being represented is easy to digest. While a visualisation like this one may be very good this example only represents a small sample of data, namely U.S Department of Energy sites. More detailed information about the tests performed
can be viewed by moving the mouse over each box. However, this information is just a list of the tests performed and provides little extra context or explanation.

Another example of an interesting visualisation of IPv6 deployment is available on the Akamai website as shown in Figure 2.3. This type of historical graph clearly shows how IPv6 traffic has changed over the time period displayed on the graph. This graph also has a breakdown by continent, but no further information and no way to directly compare the traffic between continents. IPv6Matrix has data from a much longer period of time than that shown on the Akamai website and also has this information with much greater detail.

It is important that the IPv6Matrix website should give clear visualisations of as much of the data available as possible. Historical data with a direct comparison of data sources (such as geo-located Country, TLD or Service Type) would be useful to show both overall and relative changes in IPv6 deployment over time. The Akamai website also includes links to White Papers and other resources related to IPv6, links to some information of this type would likely be beneficial on the IPv6Matrix website to enable users to find further information on IPv6.

A complete list of the websites and resources used for this research follows:
Figure 2.3: Akamai.com/ipv6

- **Cisco** - http://6lab.cisco.com/stats/
- **U.S. Department of Energy** - https://my.es.net/sites/ipv6
- **Vyncke.org** - http://www.vyncke.org/ipv6status/
- **Hurricane Electric** - http://bgp.he.net/ipv6-progress-report.cgi
- **IPv6 Observatory** - http://www.ipv6observatory.eu/indicator/ipv6-certifications/
- **v6 DEMON** - http://v6demon.ipv6observatory.eu/figures/heatmap
- **APNIC** - http://labs.apnic.net/ipv6-measurement/

2.5 Data visualisation examples

Examples of charts and graphs not related to IPv6 deployment were also researched in an effort to find suitable and visually interesting ways to display the Crawler data. These examples were found from various sources, with a large number of those found
Figure 2.4: Bubble Chart

via Javascript graph libraries and the examples used to demonstrate how the library functioned.

Figure 2.4 shows a Bubble Chart created using the D3 Javascript library. This example makes use of the size and colour of the bubbles to represent different characteristics of the data. This type of visualisation could be used on the IPv6Matrix website to show the size of domains (possibly determined by their Alexa ranking, total number of Hosts or by another means) and the type or quality of IPv6 support implemented by that domain. This chart can be very easy to understand, if appropriate information about the meaning of the size and colours of the bubbles is given, and makes it very easy to see how different aspects of the data relate to each other.

D3.geo is an extension to the D3 library providing extra functionality to assist with the creation and rendering of map data, an example of a globe projection is shown in Figure 2.5. This type of visual would be an obvious way to display certain information about the deployment of IPv6 on a country by country basis. Colour-coding countries
on a globe or map is a commonly used method to portray information about each and
gives an easy way for a viewer to rank and compare countries based on what piece of
information the colours represent. While a globe displaying IPv6 deployment informa-
tion may be compelling, it prevents the user from seeing a complete comparison of the
data and use of a flat map projection would likely be more suitable.

Figure 2.6 is a screenshot of the D3 Javascript library show reel. The show reel demon-
strates several of the types of graph and visualisation that the library is capable of
producing, including very impressive transitions between them. This demonstration is
very convincing in terms of the power and utility of the D3 library. Being able to see the
many transitions between the different types of charts also provides good insight into
how related data sets can be shown in a clear way.

A list of non-IPv6 related resources used follows:

- **D3 Bubble Chart** - http://bl.ocks.org/mbostock/4063269
- **D3 Stacked-to-Grouped Bars** - http://bl.ocks.org/mbostock/3943967
- **D3 Choropleth** - http://bl.ocks.org/mbostock/4060606
D3 Show Reel

![D3 Stock Chart](image)

**Figure 2.6: D3 Show Reel**

- **D3 Show Reel** - http://bl.ocks.org/mbostock/1256572
- **D3 Show Reel (full)** - http://vimeo.com/29862153
- **Crossfilter** - http://square.github.io/crossfilter/
- **D3 Tooltips** - http://bl.ocks.org/Caged/6476579
- **mapSVG** - http://codecyon.net/item/jquery-interactive-svg-map-plugin/full_screen_preview/1
- **Kartograph** - http://kartograph.org/
- **Stockmapper** - http://www.stockmapper.com/

### 2.6 Website technologies

#### 2.6.1 Using JavaScript

Delivering accessible, interactive experiences across the web is currently limited to only a few possibilities. These include using a proprietary browser plugin like Adobe Flash Player or the standardised ECMAScript, known as JavaScript, which is implemented in all modern browsers. Both approaches induce accessibility limitations compared to using plain HTML, but are necessary in order to produce complex client-side interaction. The clear choice in this case is to use JavaScript, which has proven to be the most popular
tool with which modern websites deliver such functionality, encompassing a wealth of open source libraries and large development community. This lays the foundation for many of the project design and technical decisions.

2.6.2 Web servers

Following the existing website research, a conclusion was met to implement a new website from scratch. The new site must meet several project specification goals, including the basic delivery of dynamic web pages and an Open Linked Data API.

The most popular choice for deploying a website is to use a LAMP (Linux, Apache, MySQL & PHP) software bundle or similar (e.g. Nginx instead of Apache). These offer wide ranging, flexible approaches for most small to medium sized websites. Apache and Nginx both offer sophisticated configuration files to allow complex site routing, that can employ PHP to dynamically generate pages on request and query a MySQL database. Such setups are very mature (in terms of software development), offering stable deployments, suitable for production purposes.

Alternative approaches include newer software projects such as Node.js, which has recently gained acceptance in mainstream software development. Though less mature, Node.js offers several features not found with traditional approaches, while still handling many concurrent connections. A driving factor for Node’s uptake is the fact it makes use of the V8 JavaScript engine as it’s core programming language. This ties in with the expanding area of front-end JavaScript development, employed for interactive and responsive websites. Such a high crossover between client-side and server-side programming reduces development time and increases cohesion between the codebases. Another advantage of using Node.js is the NPM (Node Packaged Modules Registry) which supports an open source community, developing modules to extend Node’s functionality and easily installed and managed via the npm command line tool.
Chapter 3

Design

3.1 Crawler

Various changes will be made to improve the automation and reliability of the Crawler. These updates and fixes should enable it to run without supervision and more efficiently.

To improve automation of the Crawler a better input format is needed. The Crawler should work from a single list of domains as the input rather than having a specific input file that predefined batches must be copied into, as it is currently. This means that the domain list does not need to be split into batches and ‘loaded’ in manually and the Crawler could potentially just be left to run continuously. The input list should also be updated by the Crawler at the start of each crawl, this will reduce the percentage of ‘dead’ domains being used in the data and allow new and newly popular domains to be included. The number of ‘dead’ domains can also be further reduced by maintaining a list of ignored domains that contains any domains that have been consistently unreachable for a period of time (three or six months, for example) and are no longer included in the Alexa.com list.

Another required feature of complete automation is the automatic management of the number of concurrent threads used by the Crawler while it is operating. In its current implementation a limit must be set manually in the settings file before each crawl is started, and must be set based on the next batch to be run. This feature should allow the Crawler to run by itself with a minimum of blocking and waiting time for processes.

To ensure the Crawler can operate continuously it will require increased reliability and crash recovery. In order to achieve this the Crawler should maintain information on the last completed crawl so that it can resume from the same position in the event that it crashes during operation. After each test has been completed this information will be updated.
Three scripts can be created to allow the level of automation and crash recovery required, which will be used to start, resume or stop the execution of the Crawler. The start script will cause the Crawler to begin a new crawl from the start of the input list, ignoring any information about the last test that was previously completed. Once all domains in the input have been crawled the Crawler will log the necessary information and send an email containing information about the crawl including time to complete and any domains or tests not completed for whatever reason.

A new crawl will then be started immediately following the completion of the previous one. The resume script will perform a very similar task, except it will make use of the information stored about last known position and start the Crawler from there. If information on the last known position is non-existent or indicates the end of the previous crawl, a new crawl will be started from the beginning.

A third script used to stop the Crawler will have some optional flags allowing for a ‘graceful’ end of operation. This will allow the Crawler to complete the test currently being run and update the information on crawl progress before stopping. If this flag is not used, the default action will be to immediately stop the Crawler by killing any and all processes it is using. During a crawl relevant information about the current status should be printed to stdout, such as the current test being run and current progress through the crawl, possibly with a basic estimation of time remaining until the crawl is completed.

The general operation of the Crawler can be configured by the use of a settings file, this will be extended to include more options and to also enable the execution scripts to have flags set to override or give an alternate location of the configuration file. Some potential flags that can be set include:

- `-c` –config [file] - Gives the location of the configuration file to be used.
- `-e` –email [email address] - Sets the address to which crash reports and crawl completion information is sent.
- `-eb` –enable-batch [batch] - Enable only this batch (gTLDs or ccTLDs).
- `-ed` –enable-domain [domain] - Enable this domain (tests will be run on this domain even if it is in the list of domains to ignore).
- `-et` –enable-test [test] - Enable this test (overrides the setting in the configuration file).
- `-db` –disable-batch [batch] - Disable this batch (gTLDs or ccTLDs).
- `-dd` –disable-domain [domain] - Disable this domain.
Chapter 3 Design

- `-dt -disable-test [test]` - Disable this test (overrides the setting in the configuration file).
- `-i -input [file]` - Gives the location of the input domain list to be used.
- `-l -log [directory]` - Set the directory to output log files in.
- `-n -net [number]` - Sets the maximum number of network requests that can be sent per second.
- `-t -threads [number]` - Sets the maximum number of concurrent threads to be used. This overrides the Crawler’s automatic calculation of this value.

3.2 Data processing

The overriding limitation of the existing output data is that it only uses data from the latest crawl. There are over 200 gigabytes and counting of raw data, spread over 90 different crawler runs, which have been taken since 2010 and are likely to carry on into the foreseeable future. Quite simply, there is a wealth of historical IPv6 adoption data that is being discarded. The customer, Oliver Crepin-Leblond, highlighted this in the very first meeting and expressed particular interest in being able to analyse trends in IPv6 adoption over time. Implementing this extra dimension into data analysis alongside fixing other outstanding issues requires a complete overhaul of the existing data processing scripts.

3.2.1 Database design

The raw scan data continues to use the existing structure and naming conventions for each SQLite database file, per tld and crawl date. Database files are located in `/crawls/<tld>/<timestamp>/<tld>`.

3.2.2 Analysis

Collection and conversion of data from the crawler log files into a database format and subsequent analysis of this data using the new SQLite databases are two distinct tasks, which are highly coupled in the existing solution and handled by a single python script (`log2db.py`). Another major issue was that the script only worked with the latest data, with no option to reconvert or summarise old data. When moving towards a solution that works with the complete input data set, generating summary data for many different dates, it was clear these tasks needed separating into cohesive modules allowing for increased flexibility particularly in regards to date.
Chapter 3 Design

- **getCrawls(date)** - takes a date in the format YYYY-MM-DD and finds the most recent crawl folder for every TLD as of that date, if no date is given the current date is used. Enables flexible selection of data points with accurate calculation (no missing data), potential for custom reports and dynamic resolution of time data.

- **csv2db(folders)** - takes a list of crawl folders and converts the raw CSV output of the crawler found in each into a SQLite database file, if an existing database file is found it is replaced.

- **generateSummary(folders, date)** - takes a list of crawl folders and a date, performs summary analysis queries across all SQLite databases in the given folders, inserting new summary data points at the date given into the MySQL IPv6Matrix database.

This structure allows for recalculating summaries on a snapshot of the crawler data at different times without expensive rebuilding of underlying raw data storage. Alternatively this also allows for rebuilding summaries for all crawl folders or only analysing one folder at a time, which is incredibly helpful during development. The low coupling enables parallel development of all three key modules, with separate unit testing before combining all three to fully process a new set of raw data.

### 3.2.2.1 CSV conversion

The simplest, fastest and most efficient method to convert the raw CSV output of the crawler into a queryable database format would be using SQLite’s own import functionality. However upon further investigation several critical drawbacks with this option were discovered. Firstly, the SQLite command `.import` allows for the import of csv files into a database from the SQLite command line shell, but this feature is not available via SQL commands using the python library, limiting its utility. The other limitation is that this would not allow for any custom input processing of the CSV log files such as: validation, formatting and discarding superfluous data. Therefore a python script should be written to handle this process.

A critical issue that should be resolved is correct escaping of fields being imported by using parametrized SQL queries and full utf8 Unicode support for international characters. Where possible values should be mapped to the correct value for a SQLite database, for example ‘n/a’ should be NULL in the database. Performance should be a priority - maximising insert speed into the SQLite summary databases.

Output SQLite database structure should remain as similar as possible to the existing for backwards compatibility. Extra indexes will be added on host columns in every table to allow for fast join on hosts between different tables. Table structures and indices will be defined in the `globals.py` file.
3.2.2.2 Summary generation

A large part of the development process for this stage will involve writing and optimising the SQL queries executed on the SQLite databases. All data made available with the API has a country associated with every data point, this allows the website to map all data onto a world map for many different IPv6 deployment metrics. To allow all data to be grouped by country, when generating summary data, where possible the metrics table in use should be joined with the appropriate geoip table on the host column.

For queries with large results, the fetching and subsequent insertion into MySQL should be batched to reduce peak memory usage.

SQLite Queries

- **Host Type Summary** - calculate count of IPv4, IPv6 and dual-stack enabled hosts by country
- **Ping Summary** - calculate average ping times and count faster IPv6 hosts, only for dual-stack hosts
- **Path Summary** - calculate average trace route hops and count faster IPv6 routes, only for dual-stack hosts
- **Reachability Summary** - query counting reachable IPv4 and IPv6 hosts grouped by country, run on all service reach test tables
- **Domain Summary** - generate summary hosts ping path and reachability data for every distinct domain name. It may be possible to generate DomainPenetration data from this query, without having its own separate query.

3.2.2.3 Selecting data points

An unforeseen issue until late in the design process was selecting the set of dates which should be used as data points to generate a summary ‘snapshot’ of data at that time. In a single crawler run, only a subset of the input domain list (based on a selection of TLDs) is crawled. The schedule order and exact subsets of input data follow no set pattern and have changed many times over the past 3 years of data collection. This is due to an informal crawler regime, tweaking for performance by Olivier and occasional crawler crashes. The result of this is the complication that simply choosing to generate a summary of existing data at regular intervals or every time a crawl had completed would result in repeated and biased data. The data bias would be caused by the fact that crawl subsets are not random, but chosen by TLD, which creates artificial jumps in data for an individual TLDs compared to others. For example if only the .com subset
had been crawled, a jump in IPv6 host percentage would be seen in the graphs, but it would look like every tld other than .com was stagnant in IPv6 adoption.

To overcome this issue, data points had to be carefully selected in a manner that meant all domains had been crawled at least once since the previous data point. This is not a strict rule however, just the methodology chosen to give the best representation of the existing data set, additional data points for different dates could easily be added using the generateSummary script - increasing the resolution of output data. In order to find these ‘complete re-crawl’ dates firstly a list of all 6111 crawl date sub folders (/crawls/tld/timestamp/) and their parent tld, which was then sorted by date. A sequential scan through the list stopping each time all 256 unique input tlds had been seen then produces the required list of dates. These dates ranged from 2010-08-21 to 2013-07-30 with a total of 22 data points, an average of one complete crawl every 7 weeks.

3.3 API design

3.3.1 Routes, resources & queries

3.3.1.1 IPv6 Hosts

/hosts The total number of crawled Hosts and how many of those have some IPv6 implementation.

/hosts Summary of total IPv6 Hosts by date.
Returns columns: date, hosts, hosts6.

/hosts/group-:group A more detailed breakdown by group type.
Grouped by either: country, tld or type.
Returns columns: date, hosts, hosts6, :group.

/hosts/group-:group/country-:country Grouped data for a specific country, any ISO 3166 Alpha-2 country code e.g. fr, gb, us etc.
Returns columns: date, hosts, hosts6, :group.

3.3.1.2 Dual Stack

/dualstack The total number for each Host IP implementation (IPv6, IPv6, Dual Stack, No IP).

/dualstack Summary of Dual Stack data.
Returns columns: date, hosts, hosts4, hosts6, dualstack, noip.
/dualstack/group-:group A more detailed breakdown. 
Grouped by either: country, tld or type.
Returns columns; date, hosts, hosts4, hosts6, dualstack, noip, :group.

/dualstack/group-:group/country-:country Data for a specific country. 
Any ISO 3166 Alpha-2 country code e.g. fr, gb, us etc.
Returns columns; date, hosts, hosts4, hosts6, dualstack, noip, :group.

### 3.3.1.3 Ping

/ping The average response times over IPv4 and IPv6 for Dual Stack Hosts.

/ping Summary of Ping data.
Returns columns; date, hosts, faster, ping4, ping6.

/ping/group-:group A more detailed breakdown. 
Grouped by either: country, tld or type.
Returns columns; date, hosts, faster, ping4, ping6, :group.

/ping/group-:group/country-:country Data for a specific country. 
Any ISO 3166 Alpha-2 country code e.g. fr, gb, us etc.
Returns columns; date, hosts, faster, ping4, ping6, :group.

### 3.3.1.4 Path

/path The average Traceroute length of IPv4 and IPv6 for Dual Stack Hosts.

/path Summary of Path data.
Returns columns; date, hosts, fewer, hops4, hops6.

/path/group-:group A more detailed breakdown. 
Grouped by either: country, tld or type.
Returns columns; date, hosts, fewer, hops4, hops6, :group.

/path/group-:group/country-:country Data for a specific country. 
Any ISO 3166 Alpha-2 country code e.g. fr, gb, us etc.
Returns columns; date, hosts, fewer, hops4, hops6, :group.

### 3.3.1.5 Reachability

/reach The total number of Hosts that are Reachable over each Protocol (HTTP / HTTPS / SMTPS).
Chapter 3 Design

/reach Summary of Reachability data.
Returns columns; date, hosts4, hosts6, reach4, reach6.

/reach/group-:group A more detailed breakdown.
Grouped by either; country, tld or service.
Returns columns; date, date, hosts4, hosts6, reach4, reach6, :group.

/reach/group-:group/country-:country Data for a specific country.
Any ISO 3166 Alpha-2 country code e.g. fr, gb, us etc.
Returns columns; date, hosts4, hosts6, reach4, reach6, :group.

3.3.1.6 Domains

/domains The total number of IPv6 enabled Domains.

/domains Summary of Domains data.
Returns columns; date, domains, domains6, www, ns, mx, ntp.

/domains/group-:group A more detailed breakdown.
Grouped by either; country or tld.
Returns columns; date, domains, domains6, www, ns, mx, ntp, :group.

/domains/group-:group/country-:country Data for a specific country.
Any ISO 3166 Alpha-2 country code e.g. fr, gb, us etc.
Returns columns; date, domains, domains6, www, ns, mx, ntp, :group.

3.3.1.7 Domain Search

/domain-google.com Detailed data for a specific domain and domain search functionality.

/domain-:domain All data gathered on a specific :domain by date.
Returns columns; date, country, www_hosts, www_hosts6, mx_hosts, mx_hosts6, ns_hosts, ns_hosts6, ntp_hosts, ntp_hosts6, http4, http6, https4, https6, smtp4, smtp6, faster, pingratio, fewer, hopratio.

/domain/search-:search Search for a domain starting with the string :search.
Returns a list of matched Domains.

3.3.1.8 Country Overview

/country-gb Detailed data for a specific country.
/country-:country All data gathered on for a specific :country by date. Returns columns; date, country, hosts, hosts4, hosts6, hosts6only, dualstack, noip, pinghosts, faster, ping4, ping6, pathhosts, fewer, hops4, hops6.

### 3.3.2 Data formats

The API offers JSON, XML, HTML and CSV data formats.

The data format is specified as the first identifier or as a subdomain. For example: ipv6matrix.org/json or json.ipv6matrix.org.

#### 3.3.2.1 JSON

/json returns an application/json response. All JSON responses are hierarchically structured by key, in order of the data selection defined by the URI. This simplifies data access and reduces response sizes.

For example JSON response, nested by date then country:

```json
{
  "Mon Sep 13 2010" : {
    "AD" : { "hosts" : 38, "hosts6" : 0 },
    "AE" : { "hosts" : 1087, "hosts6" : 5 },
    ...
  },
  "Sat Oct 23 2010" : {
    "AD" : { "hosts" : 38, "hosts6" : 0 },
    "AE" : { "hosts" : 1093, "hosts6" : 5 },
    ...
  }
}
```

#### 3.3.2.2 XML

/xml returns a text/xml response, with each row and column as an individual tag.

For example:

```xml
<response>
  <row>
    <date>Mon Sep 13 2010 01:00:00 GMT+0100 (BST)</date>
    <hosts>38</hosts>
    <country>AD</country>
  </row>
</response>
```
3.3.2.3 HTML

/html returns a text/html response, data is formatted into a HTML table.

For example:

```html
<row>
  <date>Mon Sep 13 2010 01:00:00 GMT+0100 (BST)</date>
  <hosts>1087</hosts>
  <hosts6>5</hosts6>
  <country>AE</country>
</row>

3.3.2.4 CSV

/csv returns a text/plain response, with column headers as the first row.

For example:

date,hosts,hosts6,country
Mon Sep 13 2010 01:00:00 GMT+0100 (BST),38,0,AD
Mon Sep 13 2010 01:00:00 GMT+0100 (BST),1087,5,AE
```
3.3.2.5 Generic request

/data attempts to return the corresponding format to the request’s Content-Type header field. If a corresponding format can not be found, a default HTML response will be sent.

For example /data/hosts/group-country will return HTML, to match your browsers default text/html Content-Type.

3.4 Website design

The aim of the website is to deliver the IPv6 data “displayed as a variety of interactive maps, graphs and charts... allowing for a high level of flexibility and detail, providing the user with a range of data filtering parameters”. Therefore, an early technical decision was made to use JavaScript in order to generate dynamic data visualisations as part of a comprehensive ‘web-app’ style project. This in turn influenced many of the core design decisions.

The design can be broken down into three distinct areas, each covering a different aspect of the process. Structure & navigation deals with the site map and what information each page displays. Data visualisations deals with how the data should be represented and intuitively interacted with from an end-user viewpoint. Finally, the user interface design involves the aesthetics and layout of the site, enabling the information to be displayed in an accessible and engaging way.

3.4.1 Structure & navigation

The first stage in designing the website was to plan the page structure, what information each page should display and how users will navigate between pages.

The website must deliver both an interactive visualisations and supplementary information about the IPv6Matrix project, how to use the site and the data API.

The core URI structure for the ‘Matrix’ web-app attempts to mimic the API URI design in a minimal and easy to navigate way. Each resource base (e.g. /hosts) is represented by an almost 1:1 mapping to a page in the web-app, showing visualisations specific to the type of data returned by the API. Specifying a country or domain will also work in a similar manor (e.g. /country-:country) but optional filtering queries will not be included in the page URI, to optimise site usability.

The remaining ‘static’ pages consist of; the ‘About’ page describing the purpose of the project and website, the ‘API’ page explaining how to access the API and what
information each resource returns and the ‘Credits’ page. The user should always be
able to navigate between the static pages, the Matrix home page and the individual
web-app pages in a consistent way.

The final site map results in a ‘flat’ URI hierarchy with all pages being relative to the
site root:

- / (Web-app) Matrix home page introducing the website.
- /hosts(/country-:country) (Web-app) API Hosts data.
- /dualstack(/country-:country) (Web-app) API Dual Stack data.
- /ping(/country-:country) (Web-app) API Ping data.
- /path(/country-:country) (Web-app) API Path data.
- /reach(/country-:country) (Web-app) API Reachability data.
- /domains(/country-:country) (Web-app) API Domains data.
- /domain(-:domain) (Web-app) API data on a domain search or specific :domain.
- /country-:country (Web-app) Overview of API data for a specific :country.
- /about (Static) Information about IPv6Matrix.org and the IPV6Matrix project.
- /api (Static) Information on how to use the API
- /contact (Static) Contact information about the project.

3.4.2 Data visualisation & interaction

To fulfil the project goal of providing interactive and detailed data views, a selection of
visual formats have been chosen with which to represent the data.

Map An underpinning visual concept for the IPv6Matrix project is to represent the
data on a map, as the project is highly linked with geographical information about IPv6
deployment around the globe. Therefore, both a ‘spherical’ orthographic (Figure B.7)
and flat ‘mercator’ projection (Figure B.8) of the world have been chosen. The matrix
home page will display the globe that can be interacted with and rotated as an engaging
introduction to the website. The remaining pages will feature a flat projection, aimed
at displaying all available data at once. This will also enable the user to zoom, pan and
select individual countries for a detailed breakdown.
**Graphs** In order to display more complex data representations over time, a series of graphs will be incorporated showing relevant comparisons between data sets for each page (Figure [B.9]). Each graph will follow a pattern of displaying the date range along the x axis and one or more data metrics as either a line or stacked area chart along the y axis. Graphs that need to display a large number of different metrics will use a tabbed button paradigm to select the currently displayed metric. In addition, when the user ‘hovers’ over a graph an interactive tooltip will be display, showing data for a specific date.

**Table** A comprehensive table will show the original data and metric calculations that all other visualisations on the page use (Figure [B.10]). It will also feature the ability to sort the data via each column value and display aggregated statistics (e.g. total number of hosts, average number of hosts).

### 3.4.2.1 Data selection

Empowering users to have complete control over the displayed data is crucial to the functionality of the website. At the most basic level a primary menu bar for the web-app will allow users to select the page / displayed data resource (Figure [B.1]).

However, for more fine grained options a centralised controller is needed (Figure [B.3]). This will incorporate the use of a date selection ‘slider’ paradigm that users can ‘click and drag’ to manipulate the currently displayed date on the map and table (Figure [B.4]). A second setting represented by a ‘drop-down menu’ will enable users to select the current grouping of the data (e.g. by country, TLD, service type etc.) that will update the table and appropriate graphs.

Finally, more advanced filtering parameters must be made available, to make full use of the underlying data API functionality. These will include the option to select a minimum data sample size and to filter data by service type and TLD (Figure [B.5]).

### 3.4.3 User Interface

The UI design focuses on the core aesthetics of the site and the element display structure. The website design must follow the overarching goals of the project, to make data easy to access and the website appealing to use, striking a balance between providing intuitive interface interaction, while offering a sophisticated range of data selection and filtering options.

The interface went through several design iterations before the final layout was settled upon. The first prototype, Figure [B.1] depicts many of the core interface elements that
can be seen in the final design. Here the focus was on what visualisations each page would show and how this would tie in with the user selection settings.

![Wireframe prototypes 1](image)

**Figure 3.1:** Wireframe prototypes 1

After an initial development stint to create a prototype website for Progress Seminar 1, a further iteration of the design was constructed, seen in Figure 3.2. It consists of 4 core areas; the header, menu bar, sidebar and main content area and focuses more on the presentation of the data towards the end-user.

The header is omnipresent throughout the site and displays the site logo along with the list of static pages for consistent navigation, while the menu bar allows for navigation between pages of the web-app. Influence for the menu bar styling was taken from the Google Analytics website, which inspired some other design decisions.

The addition of the sidebar from the first wireframe set, is to summarise the information for the page and direct users to important links e.g. the current data API URI. This enables a wide audience to engage with the site, by explaining what data the current page is displaying, how it was calculated and also basic technical concepts (e.g. what a ‘host’ is). This took styling influence from the Backbone API website, which establishes a fixed, scrolling side bar.

While organising the interface and information presented between pages, the website’s target audience was constantly being considered. In order to be accessible to the general
public the site opens with an interactive globe and a short introduction to the IPv6Matrix project. The following pages order each visual representation from least to most specific, as the user scrolls down the main content area; starting with the map to the selection of graphs and ending with the comprehensive data table. Moving through the website continues this idea, as data and information become more specific, catering for more technical users who may require advanced data breakdowns and filtering parameters.

Throughout the design process, space between data representations was considered, so that users are not overwhelmed by information, unlike some examples researched in Section 2.4. Titles for the descriptions, graphs and table help separate the information in both the sidebar and main content area. In addition to this, a tooltip paradigm is integrated across the interface, providing detailed explanations for each data representation and even each column in the table. These are only displayed when the user ‘hovers’ over a question-mark icon, so as to reduce the initial bombardment of information.

The last and most crucial concept of the user interface is the centralised controller. This also went through a number of design phases. It was found that having the slider sitting in with the visualisation content area made it difficult to access if it was off screen (as in Figure 3.1b). Therefore, the group settled on having the controller fixed to the bottom of the browser window, aligned to the main content area (Figure 3.2b controller). In this way, it is always visible and accessible to the user, while indicating that it is selecting the currently displayed data subset. Again, more advanced options are initially hidden, but can be quickly shown by clicking the settings ‘cog’ icon.
<table>
<thead>
<tr>
<th>Table</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>MX_tld</td>
<td>[type, domain, host, ipv4, ipv6, rank]</td>
</tr>
<tr>
<td>NS_tld</td>
<td>[type, domain, host, ipv4, ipv6, rank]</td>
</tr>
<tr>
<td>NTP_tld</td>
<td>[type, domain, host, ipv4, ipv6]</td>
</tr>
<tr>
<td>WWW_tld</td>
<td>[type, domain, host, ipv4, ipv6]</td>
</tr>
<tr>
<td>domainPenetration_tld</td>
<td>[domain, ns, mx, www, ntp]</td>
</tr>
<tr>
<td>geoip_MX_tld</td>
<td>[type, domain, host, ipv4, ipv6, asn, city, region_name, country_code, longitude, latitude]</td>
</tr>
<tr>
<td>geoip_NS_tld</td>
<td>geoip_NTP_tld</td>
</tr>
<tr>
<td>geoip_WWW_tld</td>
<td>ip6Type_MX_tld</td>
</tr>
<tr>
<td>ip6Type_NS_tld</td>
<td>[type, domain, host, ipv6, valid, prefixid, ipv6type]</td>
</tr>
<tr>
<td>ip6Type_NTP_tld</td>
<td>[type, domain, host, ipv6, valid, prefixid, ipv6type]</td>
</tr>
<tr>
<td>ip6Type_WWW_tld</td>
<td>[type, domain, host, ipv6, valid, prefixid, ipv6type]</td>
</tr>
<tr>
<td>path_MX_tld</td>
<td>[type, domain, host, ipv4, ipv6, mtu4, hops4, back4, path4, mtu6, hops6, back6, path6]</td>
</tr>
<tr>
<td>path_NS_tld</td>
<td>[type, domain, host, ipv4, ipv6, mtu4, hops4, back4, path4, mtu6, hops6, back6, path6]</td>
</tr>
<tr>
<td>path_NTP_tld</td>
<td>[type, domain, host, ipv4, ipv6, mtu4, hops4, back4, path4, mtu6, hops6, back6, path6]</td>
</tr>
<tr>
<td>path_WWW_tld</td>
<td>[type, domain, host, ipv4, ipv6, mtu4, hops4, back4, path4, mtu6, hops6, back6, path6]</td>
</tr>
<tr>
<td>ping_MX_tld</td>
<td>[type, domain, host, ipv4, ipv6, count, min, avg, max, std, min6, avg6, max6, std6]</td>
</tr>
<tr>
<td>ping_NS_tld</td>
<td>[type, domain, host, ipv4, ipv6, count, min, avg, max, std, min6, avg6, max6, std6]</td>
</tr>
<tr>
<td>ping_NTP_tld</td>
<td>[type, domain, host, ipv4, ipv6, count, min, avg, max, std, min6, avg6, max6, std6]</td>
</tr>
<tr>
<td>ping_WWW_tld</td>
<td>[type, domain, host, ipv4, ipv6, count, min, avg, max, std, min6, avg6, max6, std6]</td>
</tr>
<tr>
<td>reverse_MX_tld</td>
<td>[type, domain, host, ipv4, ipv6, name4, name6]</td>
</tr>
<tr>
<td>reverse_NS_tld</td>
<td>[type, domain, host, ipv4, ipv6, name4, name6]</td>
</tr>
<tr>
<td>reverse_NTP_tld</td>
<td>[type, domain, host, ipv4, ipv6, name4, name6]</td>
</tr>
<tr>
<td>reverse_WWW_tld</td>
<td>[type, domain, host, ipv4, ipv6, name4, name6]</td>
</tr>
<tr>
<td>soa_NS_tld</td>
<td>[type, domain, soa, primary_by_rank, primary_inhouse, secondary, total, contact, serial, refresh, retry, expire, minimum]</td>
</tr>
<tr>
<td>tcp25_MX_tld</td>
<td>[type, domain, host, port, ipv4, ipv6, tcp, tcp6]</td>
</tr>
<tr>
<td>tcp443_WWW_tld</td>
<td>[type, domain, host, port, ipv4, ipv6, tcp, tcp6]</td>
</tr>
<tr>
<td>tcp80_WWW_tld</td>
<td>[type, domain, host, port, ipv4, ipv6, tcp, tcp6]</td>
</tr>
<tr>
<td>tls_MX_tld</td>
<td>[type, domain, host, ipv4, reachable, tls]</td>
</tr>
</tbody>
</table>
Chapter 4

Technical Approach

4.1 Data processing

The IPv6 Matrix crawler generates CSV log files, nested in a timestamp folder under a TLD folder. These must be discovered by date and converted into a SQLite database file to allow for easy analysis using SQLite. The resulting summary data is to be stored in a MySQL database for optimised delivery via a data API frontend.

4.1.1 CSV conversion

The simplest, fastest and most efficient method to convert the raw CSV output of the crawler would be using SQLite’s own import functionality. However, upon further investigation several critical drawbacks with this option were discovered. Firstly, the SQLite command \texttt{.import} allows for the import of CSV files into a database from the SQLite command line shell, but this feature is not available via SQL commands using the python library, limiting its utility. The other limitation is that this would not allow for any custom input processing of the CSV log files such as: validation, formatting and discarding superfluous data.

The python script uses the following key modules:

- \texttt{os} - For browsing the filesystem for log files, eg. \texttt{os.listdir()}
- \texttt{csv} - Opening and parsing log files line by line using \texttt{csv.reader()}
- \texttt{sqlite3} - SQLite driver to create SQLite database files and perform SQL queries upon them.

For each TLD crawl folder (/crawls/tld/timestamp/), a new database file named after the current TLD \texttt{tld.db} is created. For each .log file found in this directory, a database
Chapter 4 Technical Approach

Table is created using the schemas defined in `globals.py`. The python module `csv` is used to read the CSV file line by line in universal newline mode to avoid line parsing errors. The CSV delimiter is set to comma as would be expected, but unfortunately this causes some issues when commas are erroneously included in DNS records. This cannot be avoided on existing data as the records were not quoted or escaped, so in order to prevent erroneous data entering the SQLite database row length checks are carried out.

To save file space and increase performance, superfluous rows from the input data are ignored if they have no data in ‘required’ columns. For example, in tables such as `ip6Type`, there is nothing gained by keeping an entry for every host even if the host has no IPv6 record.

Individual fields are parsed using utf8 Unicode strings to ensure correct support of all international characters in domains and DNS records. Unicode support fixes existing data processing bugs when combines with SQL prepared statements as the string is passed as a parameter to the SQL driver and does not need to be escaped, ensuring no input characters are lost or break queries. Fields are checked against a value conversion mapping defined in `globals.py`, which converts values into the correct type for SQLite, for example the string ‘n/a’ is replaced with the python `None` type. Indexes are added as defined in `globals.py`, `ANALYSE` is then executed to calculate index metrics such as cardinality and coverage to optimise SQLite’s usage of these indexes.

4.1.1.1 Performance optimisation

The primary optimisation for the conversion process is SQLite inserts per second, this is vital for importing a large amount of data into a database, which is essentially bulk insertion.

Instead of inserting rows as they are read line by line from the CSV file and processed `connection.executemany` is used to perform all inserts for a database table with a single prepared SQL statement. The improvement can be seen simply from the SQL requests required alone:

```
1  INSERT INTO table (a,b,c) VALUES (?,?,?); parameters = (1,2,3);
2  INSERT INTO table (a,b,c) VALUES (?,?,?); parameters = (4,5,6);
3  INSERT INTO table (a,b,c) VALUES (?,?,?); parameters = (4,5,6);
```

compared to `connection.executemany`:

```
1  INSERT INTO table (a,b,c) VALUES (?,?,?); parameters = ((1,2,3), (4,5,6),
(7,8,9));
```

The volume of string and statement processing performed by the SQLite driver is drastically reduced in addition to allowing further internal optimisations as the driver does
not need to perform multiple query and transaction initialisation, clean up and integrity operations. Along with prepared SQL statements themselves, this was one of the key performance oversights with the old csv2db system.

Several SQLite specific settings are changed to improve performance by at least two orders of magnitude in terms of inserts per second, these are configured using SQL PRAGMA statements:

- **PRAGMA synchronous = OFF** - Disables waiting for data to be written to disk and validating its integrity before allowing any further queries. A disk flush and check for every insert slows inserting thousands of rows by several orders of magnitude. The only risk with this method is that sudden power loss during SQL statements that modify the database could result in database corruption (in practice, you are more likely to suffer a catastrophic disk failure or some other unrecoverable hardware fault). This risk does not apply to our situation, as sudden power loss mid-conversion would result in the conversion process to be restarted from scratch and corruption would not result in any actual data loss. Olivier has experienced an unusually high rate of disk failures on the machine which performs this import operation and **PRAGMA synchronous = OFF** should reduce disk thrashing and allow full use of main memory as a disk cache during bulk imports, which could potentially help solve the drive failure issue.

- **PRAGMA journal_mode = OFF** - Disables transaction handling and roll-back functionality as they are not needed for the import process and add extra overhead.

The database indexes are only added after all data is inserted into a database. This allows the B Tree index to be built via ‘bulk loading’. Although each insert has a $O(\log\cdot n)$ overhead, giving $O(n \cdot \log\cdot n)$ and bulk loading requires sorting the keys before building an index which is $O(n \cdot \log\cdot n)$ bulk loading is still significantly faster due to the I/O overhead of actual page reads and constantly splitting leaves when inserting into the tree. An additional benefit is that the resulting freshly built B-Tree structure will be perfectly balanced and stored sequentially in memory, giving maximum possible search and ranged result efficiency with minimal random I/O. Normally a major issue with this approach is that index rules such as unique columns cannot be applied - these are not required for validation or optimisation given the type of data and queries used for data analysis, therefore this may be safely overlooked.

### 4.1.2 Summary generation

All data made available with the API has a country associated with every data point, this allows the website to map all data onto a world map for many different IPv6 deployment metrics. In order achieve this, data must be grouped by country when
performing analysis. This means that every SQL summary query on the crawler output must perform a join between the statistical data and the geoip database tables that contain geolocation information for every host. During initial unit and performance testing of the SQLite queries (on a subset of the full data) adding an index to the host column on all tables allowed this join to take place within an acceptable time (less than 1 second for 100,000 hosts). However, in later stages of development the performance catastrophically degraded during these joins. Upon further investigation this was due to the host index cardinality; instead of a host-name being unique within a table, there could be up to 600,000 duplicate entries for name servers in particular - as millions of domain names may use the same name server. The result size of a SQL join is approximately \( nm \) where \( n \) and \( m \) are the average cardinality of each column (in this case \( n \approx m \) so the result size is \( n^2 \)). There are 3,700,000 .com name servers, but the average name server is repeated 20,000 times giving a join result of \( 20,000^2 = 400,000,000 \) rows instead of the expected 3,700,000. Not only is this much slower to perform but also results in incorrect data.

To resolve this issue several options were investigated such as sub queries restricting geolocation data to be unique by host, but the optimal solution was to generate new geoip tables. These tables only include unique host names which have location data (removing superfluous columns such as domain, ipv4 and ipv6) with an additional count column. With the addition of an index, this allows for very fast lookup of a particular host’s location. These tables are named `geoip_sum_MX_tld`, `geoip_sum_NS_tld`, `geoip_sum_NTP_tld` and `geoip_sum_WWW_tld`. The full geoip tables are kept for backwards compatibility and simplification of the IPv6 penetration and host type summary queries, as all data they need can be found in the geoip tables. Only the country code column is used, as it was decided at the design stage there would be no real benefit to higher geographical resolution in the API output.

**Host Type Summary** Initially this query used the `MX_tld`, `NS_tld`, `TP_tld` and `WWW_tld` tables, however, all the information needed (list of hosts and their IPv4 or IPv6 addresses) can be found in the full geoip tables. This saves a join and makes the query nearly instant, even with millions of hosts. As described in the previous section, on CSV conversion the ‘n/a’ strings output by the crawler when an A or AAAA record was not found are replaced with NULL values which allows for efficient NULL and NOT NULL comparisons compared to string comparison. An additional benefit is closer adherence to SQL standards and improved code readability. Note that in this query and the MySQL database `hosts6` is the count of IPv6 only hosts.

```sql
SELECT ifnull(g.country_code, 'ZZ') as country,
COUNT() as hosts,
SUM(g.ipv4 NOT NULL AND g.ipv6 IS NULL) as hosts4,
SUM(g.ipv4 IS NULL AND g.ipv6 NOT NULL) as hosts6,
SUM(g.ipv4 NOT NULL AND g.ipv6 NOT NULL) as dualstack,
```
P ping summary - Only hosts that were successfully tested for ping on both IPv4 and IPv6 are included in the statistics, by checking that both the average IPv4 ping and IPv6 ping is set for each host. \textsc{Total} is used instead of \textsc{sum} throughout many of these queries, \textsc{total} is a custom SQLite SQL command which always returns a float in contrast to the standard \textsc{sum} which returns an integer, or NULL when only given NULL values to sum. It is used to sum average pings as there is potential for all values in a ping column having a NULL value for a group. This calculated an average IPv6 ping of 0 instead of a NULL value when there were no IPv6 hosts in a country.

\begin{verbatim}
SELECT COUNT() as pinghosts,
SUM(p.avg>p.avg6) as faster,
TOTAL(p.avg) as ping4,
TOTAL(p.avg6) as ping6,
ifnull(g.country_code, 'ZZ') as country
FROM geoip_sum_0 as g, ping_0 as p
WHERE g.host = p.host
AND avg NOT NULL
AND avg6 NOT NULL
GROUP BY country_code;
\end{verbatim}

Path Summary - This query is very similar to the ping summary SQL, with hops instead of ping.

\begin{verbatim}
SELECT COUNT() as pathhosts,
SUM(p.hops4>p.hops6) as fewer,
TOTAL(p.hops4) as hops4,
TOTAL(p.hops6) as hops6,
ifnull(g.country_code, 'ZZ') as country
FROM geoip_sum_0 as g, path_0 as p
WHERE g.host = p.host
AND hops4 NOT NULL
AND hops6 NOT NULL
GROUP BY country_code;
\end{verbatim}
Reachability Summary  Both **TOTAL** and **COUNT** are used in this query, as a host count for IPv4 or IPv6 only needs to count NOT NULL values whereas the reachable columns can be True, False or NULL and only True values should be counted.

```sql
1 SELECT ifnull(g.country_code, 'ZZ') as country,
2 COUNT(s.ipv4) as hosts4,
3 COUNT(s.ipv6) as hosts6,
4 TOTAL(s.tcp) as reach4,
5 TOTAL(s.tcp6) as reach6
6 FROM geoip_sum_{0} as g, {1}_{0} as s
7 WHERE g.host = s.host
8 GROUP BY country_code;
```

**Listing 4.4:** SQL summary query counting reachable IPv4 and IPv6 hosts grouped by country, run on all service reach test tables

Domain Summary  Both domains and domain penetration data is calculated with a single query. Fetching of results and subsequent insertion into MySQL is batched into a maximum of 10,000 rows at a time for domains data to keep memory usage down, particularly for `.com` as there are over half a million domain names in this set. This keeps total memory usage of both MySQL and python to under 1 gigabyte.

```sql
1 SELECT d.domain,
2 ifnull((SELECT geoip_sum_WWW_{0}.country_code FROM geoip_sum_WWW_{0}
       WHERE geoip_sum_WWW_{0}.host='www.'||d.domain), 'ZZ') as country,
3 (SELECT COUNT(*) FROM WWW_{0} WHERE WWW_{0}.domain=d.domain) as www_hosts,
4 (SELECT COUNT(*) FROM WWW_{0} WHERE WWW_{0}.domain=d.domain AND ipv6 NOT NULL) as www_hosts6,
5 (SELECT COUNT(*) FROM NS_{0} WHERE NS_{0}.domain=d.domain) as ns_hosts,
6 (SELECT COUNT(*) FROM NS_{0} WHERE NS_{0}.domain=d.domain AND ipv6 NOT NULL) as ns_hosts6,
7 (SELECT COUNT(*) FROM MX_{0} WHERE MX_{0}.domain=d.domain) as mx_hosts,
8 (SELECT COUNT(*) FROM MX_{0} WHERE MX_{0}.domain=d.domain AND ipv6 NOT NULL) as mx_hosts6,
9 (SELECT COUNT(*) FROM NTP_{0} WHERE NTP_{0}.domain=d.domain) as ntp_hosts,
10 (SELECT COUNT(*) FROM NTP_{0} WHERE NTP_{0}.domain=d.domain AND ipv6 NOT NULL) as ntp_hosts6,
11 (SELECT COUNT(*) FROM tcp80_WWW_{0} WHERE tcp80_WWW_{0}.domain=d.domain AND tcp) as http4,
12 (SELECT COUNT(*) FROM tcp80_WWW_{0} WHERE tcp80_WWW_{0}.domain=d.domain AND tcp6) as http6,
13 (SELECT COUNT(*) FROM tcp443_WWW_{0} WHERE tcp443_WWW_{0}.domain=d.domain AND tcp) as https4,
14 (SELECT COUNT(*) FROM tcp443_WWW_{0} WHERE tcp443_WWW_{0}.domain=d.domain AND tcp6) as https6,
15 (SELECT COUNT(*) FROM tcp25_MX_{0} WHERE tcp25_MX_{0}.domain=d.domain AND tcp) as smtp4,
```
Chapter 4 Technical Approach

4.2 Node.js web server

Node.js is the software package of choice with which we deliver both the website and data API via a single web server instance over HTTP. This entails delivering a small selection of semi-static web pages for the website and returning formatted data queried from a database for the data API.

All routing, file access and database querying is processed with only a single JavaScript application, in contrast to traditional LAMP software bundles where this is usually defined across several configuration files and PHP scripts. However, Node.js natively supports CommonJS to ‘require’ external files, allowing the application to be spread across multiple file ‘modules’ e.g. (web.js and data.js routing files).

The large cross-over between the Node server and web-app implementation reduces development time and maintains a high level cohesion between the two codebases. The web server is built in conjunction with several NPM modules, most notably the Express framework and the Jade template engine. These provide common features for routing and rendering web pages, assuring optimised code and improved development speeds.

As Node is essentially an environment for executing JavaScript, it is also sometimes used as a one-time utility, without running it as a persistent web server.

4.2.1 Express middle-ware

Express is a framework that provides HTTP utility methods as a robust and user-friendly API. Built on the Connect middleware server framework, it makes available a selection of features on top of the basic Node server; HTTP caching, content negotiation,
environment based configurations and more. Its main use in the project is to define each URI route with the appropriate response and provides easy integration with the Jade template engine.

The web server is configured separately for development and deployment purposes, based on the node environment variable (\texttt{NODE\_ENV}). When set to \texttt{dev}, the server runs on port 3000 and delivers all static files from the /\texttt{dev} directory. In normal operation the server listens on HTTP port 80 and delivers a minimised version (Section 4.3.6) of the static files from /\texttt{deploy}.

Express is configured to minimise all plain text responses, where white-space formatting is unnecessary. It also, automatically manages \texttt{Last-Modified} header files, returning an empty HTTP 304 response if the file is unchanged, ensuring reduced server transmission rates.

An Express code example highlights a convenient way to define complex routes with overlapping resource identifiers, using a recursive function, as in Listing 4.6. This technique is used in both the data and web routing and works by iterating over the keys of an object, appending them together, while adding any response functions to the Express routing table.

```javascript
var app = express();

// recursive map function
app.map = function(a, route){
  route = route || '';
  for (var key in a) {
    switch (typeof a[key]) {
      case 'object':
        // recurse and append to route string
        app.map(a[key], route + key);
        break;
      case 'function':
        // add route and response to the server routing table
        app[key](route, a[key]);
        break;
    }
  }
}

// route map definition
var routes = {
  '/:format(data|json|xml|html|csv)': {
    '/hosts': {
      '/group-:group(tld|country|type)': {
        '/country-:country': {
          get: data.v6hosts
        }
      }
    }
  }
},
```
Chapter 4 Technical Approach

Listing 4.6: Route map example for Data API with Express

The website routes are defined in the same way as Listing 4.6, but the routes for the web-app are placed in an external JSON file. This allows the same definitions to be used by both the web server and integrated into the web-app, so any modifications are reflected in both. The web server will map all web-app specific routes to the index page (/) while the web-app will listen to URI changes internally (client-side), dynamically displaying each data resource view (Section 4.3.4).

A secondary middleware called express-subdomains is also implemented to allow the routing of explicit subdomains correctly. Data API format specifiers can be used as subdomains for convenience e.g. http://json.ipv6matrix.org/hosts, while all other subdomains will route to the default home page.

4.2.2 Jade template middle-ware

Jade is a popular, high performance template language designed for rendering Extensible Markup Language (XML) documents. The Jade middleware supplies Node with the engine to parse and render .jade files, which are then minimised, sent and cached by Express.

The Jade language allows for rapid construction of XML documents by eliminating the need to write opening and closing tags, making it much more efficient for scripting documents like HTML or XML compared to a using PHP script. Jade templates accept an options object to dynamically generate sections of the document, Listing 4.7. The object is passed to the template via the render function and is accessible from in-line
JavaScript. This functionality ties in closely with the Node environment and the Express framework, making for seamless integration.

Both the website and API make extensive use of generating Jade templates with dynamic objects. The website is passed a list of static pages (navLinks in Listing 4.7), a list of web-app menu links and a list of all TLDs for the generation of the advanced settings dialog list.

```
<! -- Jade template -->
<! -- navLinks passed in from Node via Jade render engine -->
!!!
head
body
  h1 Hello world!
  ul#menu
    - navLinks.forEach(function(link){
      - var isActive = (link.label === section ? 'active' : '');
      li(class=isActive)
        a(href=link.path data-bypass)= link.label
    - });
  .content
<! -- Rendered HTML output -->
<! doctype html>
<html>
  <head></head>
  <body>
    <h1>Hello world!</h1>
    <ul id="menu">
      <li class="active">
        <a href="/" data-bypass="">matrix</a>
      </li>
      <li>
        <a href="/about" data-bypass="">about</a>
      </li>
      <li>
        <a href="/api" data-bypass="">api</a>
      </li>
    </ul>
    <div class="content"></div>
  </body>
</html>
```

Listing 4.7: Example Jade template web page

Jade templates can also extend one another, reducing code repetition. The website pages each have an individual .jade template containing their unique page content. These then all extend layout.jade which defines the generic outer page elements e.g. header, sidebar, css sources etc. Being able to modularise the code so efficiently, means
changes to the site layout are reflected on all pages, maintaining aesthetic and interactive consistency across the website.

Finally, the API also makes use of Jade to render the XML and HTML formats. The templates iterate over the returned database query object, outputting each row’s key and value in the respective format. In the example XML format output, Listing 4.8, every key in each row of the JSON object is output as `<key>value</key>`.

```javascript
if json && json.length
- var columns = [];
each val, key in json[0]
  if (json[0].hasOwnProperty(key))
    - columns.push(key);
response
each obj, row in json
row
  each key in columns
    #{key} #{json[row][key]}
```

Listing 4.8: data API XML format Jade template

### 4.2.3 Data API implementation

The data API routes are defined using a JSON tree structure for ease of use as defined in listing Listing 4.6. This is then flattened to give a list of all possible routes and their mapping to query building functions.

Initially queries were built with basic string manipulation, however this eventually resulted in messy, hard to understand code with large amounts of nested string searching. To improve the the node-sql library is used. It is a SQL query builder which allows for programmatically building a query with chainable functions, following classic node style.

```javascript
var t = db.domains;
var query = t.select()
  .where(t.domain.equals(req.params.domain))
  .order(t.date)
  .toQuery();
```

Listing 4.9: Example of building a SQL query in javascript using node-sql

Jade templates are used to format output data as HTML XML and CSV. For JSON a more advanced recursive function is used to nest data based on its grouping to minimise repeated data in the output. This also makes the JSON object simpler to navigate and lookup items.
The standard node mysql driver is used for connection over a TCP socket to the MySQL database. The connection between the Data API and the MySQL server makes use of the connection pool functionality available in the node-mysql module. This enables the Data API to automatically reconnect to the MySQL server if connection problems occur. This issue is unlikely to happen often, if at all, since the Data API and MySQL server are run on the same machine. Prior to implementation of the connection pool a single persistent TCP connection was used by the Data API. The possibility of connection issues on this link was only discovered during development outside of the local network of the test MySQL server. The connection to the MySQL server then had to be maintained using home broadband, however this caused the Data API to regularly time-out resulting in a fatal connection error that could only be remedied by restarting the Data API.

Node-mysql connection pools maintain by default up to 10 TCP connections. When the Data API queries the MySQL server a connection is requested from the pool and after the query has been completed the connection is released back to the pool. If a connection in the pool encounters a connection error that would previously have been fatal a new connection is created to replace it, allowing the Data API to function as normal even after connection errors. This also allows multiple users of the API at the same time, with the possibility of increasing the limit on the number of connections if this is required during periods of heavy load. The connection pool creates connections lazily, meaning that only the maximum concurrently requested connections up to the limit set in the configuration are ever created. As a result the API is unlikely to ever have more than one connection in the pool unless the website is under heavy load.

4.3 JavaScript web-app

Developing the JavaScript web-app was the most involved aspect of the project, spanning a large number of disciplines, coding practices and code dependencies. These dependencies can be broken down into two categories; Frameworks which enforce re-usable design patterns and Libraries which are simply a collection of associated functions or utilities.

4.3.1 Frameworks

Two core frameworks formed the basis of the web-app. These were decided upon before any development work had taken place and shaped the structure of the JavaScript application.

**Backbone.js** is a light-weight MV* framework. That is, it defines specific Model and View classes but has no dedicated Controller class, unlike the tradition MVC design pattern. This gives developers a high degree of flexibility when working with Backbone,
but also enough rope to hang oneself with. Backbone incorporates a number of other classes such as the Router and Collection classes and a extra modules, usable in any object, such as the Events module. It has one dependency, the Underscore.js utility library (Section 4.3.2), which is also used independently throughout the project.

Class definitions are loosely defined and can be used in any combination seen fit. However, each class owns utilities enhancing their basic functionality. Models have the ability to fire events when ‘attribute’ parameters are \texttt{set}(), allowing other objects to listen to these changes. Views correspond to HTML Document Object Model (DOM) elements and have integration with jQuery if it’s in use, with semi-automatic event delegation functionality. Routers have the ability to listen to and capture URI changes without reloading the current page using the HTML5 History API.

The web-app is built on the MV* design principle, whereby Models contain and calculate application state (including data loaded from the Data API), while Views listen to changes in the Model state and update accordingly. Although there are no dedicated Controllers, many of the controller type operations are included in View objects, while tradition display information is defined elsewhere in the DOM and CSS. Decoupling the DOM structure and CSS design decisions from the JavaScript is crucial for maintaining a stable application, as the browser environment is unpredictable at the best of times. In this way, Views become controllers that handle the creation and destruction of child Views based on listening to user input, Model and Router events. The MV* implementation decisions are further discussed in Section 4.3.3.

\textbf{RequireJS} is an Asynchronous Module Definition (AMD) API for loading JavaScript files and modules. It encapsulates JavaScript code in singular files, which can then call \texttt{require}() to load in further JavaScript files (similar to \texttt{import} in other languages like Python). This provides the ability to modularise JavaScript code across multiple files for large web-app projects. RequireJS actually specifies a strict procedure for ‘defining’ and ‘requiring’ modules so as to optimise the loading of external files at run-time in the browser.

The idea of modules in RequireJS coincides closely with classes in Backbone, enabling a design paradigm for defining one Backbone class per RequireJS module. This concept modernises JavaScript away from being one monolithic code file, where code must adhere to strict namespace conventions, in order to prevent variables and scopes from overriding one another. Using RequireJS mitigates this problem, as each module is automatically encapsulated in a unique namespace.

RequireJS can also be utilised with Node.js for minimising an application using \texttt{r.js} (the RequireJS optimiser). This compresses large, distributed projects into one optimised file by appending the code, removing white-space and renaming locally scoped variables to
Chapter 4 Technical Approach

one letter characters. The result is an entire web-app and its dependencies delivered in a single JavaScript file and with a reduced pay-load size.

With these two frameworks, the foundation is laid for building sophisticated applications in JavaScript. During production, code is modularised and debugging errors are scoped to specific files. For deployment, the RequireJS optimiser has many options for minimisation of the JavaScript application and corresponding files like CSS (Section 4.3.6).

4.3.2 Libraries

During development, third party libraries were constantly being considered for inclusion in the codebase. A balance between size and utility of each library was struck, favouring libraries with lightweight footprints and well defined sets of features.

**jQuery** is the most common JavaScript library, providing DOM manipulation, event handling, animation and AJAX functions through an easy-to-use and extensible API. It is tightly integrated with Backbone (though not mandatory) and is used as the primary method for specifying View objects’ corresponding DOM components. Backbone’s event module also makes use of jQuery selectors and returns jQuery event handler objects. The jQuery ‘document ready’ listener, simply `$\{(function()\{ .. \});\}$`, initialises the entire web-app once the full DOM and external resources have been loaded by the client.[22]

A criticism often made of jQuery is the trade-off between the selection engine’s execution speed and its ease-of-use, compared to using the native JavaScript selection API. This can adversely impact application speed when performing many complex DOM element selections. However, its use in the project is for one-off selections which are calculated and then stored as a variables or cached internally by Backbone, incurring negligible execution overhead. Furthermore, many other libraries also depend explicitly on jQuery, making it a mandatory requirement for most JavaScript projects.

**jQuery UI** extends jQuery with reusable User Interface components, additional animation effects and ‘widgets’. These are provided separately from jQuery as they are seen as extraneous features that would ‘bloat’ the core library. The project uses very few of the jQuery UI features, as it is mainly included as a dependency for jQuery Tree. Luckily, it can be custom built, including only features that are absolutely necessary, vastly reducing file size.[23]

**jQuery Tree** provides a drill-down, check-box tree UI component, for the advanced data filtering options panel, Figure B.11. Many similar libraries were considered, however jQuery Tree was the most lightweight and feature specific, although it requires both jQuery and jQuery UI libraries to function.[8]
RequireJS Text is a very minimal plugin for RequireJS, enabling text resource to be loaded into JavaScript in a similar way to modules. The web-app uses this extensively for including static JSON files into the code e.g. the map topography data, the web-app menu links, country code and name mappings, the TLD list etc. An additional benefit is that these files are embedded into the application when minimised, reducing size and containing it to a single file. [30]

Underscore A utility library that provides many python like array, object and string manipulation tools. [36] [33]

D3 DOM selection and manipulation library for building data driven JavaScript websites. This is used for the map projections and data display within them. [7]

tablesorter Adds interactive layer to all tables shown on the website, allowing sorting to be applied to any column. Automatically handles many different data types. [4]

nvd3 All the graphs shown on the website are produced and rendered using the NVD3 Javascript library. NVD3 is built on top of the D3 library and attempts to simplify the SVG chart and graphing functionality it contains. This allows the creation of visually impressive and interesting charts and graphs without having to delve into SVG creation and editing and individually creating each SVG element required for a chart. While this is very useful, the library is still in the relatively early stages of development and is particularly lacking in documentation. Any issues arising with NVD3 typically involved searching through the extensive source code to determine what the error was and how to fix it. [29]

Typeahead A highly configurable and flexible twitter auto complete search box library. Features multiple data sources, caching and advanced display formatting. [35]

4.3.3 MV* implementation

An MV* scheme was proposed before major development took place. It aimed at encapsulating and processing the represented data in an elegant way, shared across the application and updated cohesively via routing and user-input events. The application also requires efficient adding, removing and updating data visualisations between pages.

When preparing to develop large software engineering projects, care must be taken not to introduce a high level of coupling between modules of code. The Backbone event listening paradigm reduces a potentially large proportion of inter-class coupling, with
the overall framework consisting of 3 Models, the Router, the main View and a set of child Views. This section discusses the organisation of code and relationships between classes. More specific implementation details are discussed in Sections 4.3.4 and 4.3.5.

4.3.3.1 Models

viewModel

The viewModel is a centralised Model for storing global application state as is named so because it is passed to every View on initialisation. It keeps track of the current page location, API base URI, selected country or domain and the currently displayed date among other attributes. Changes to these parameters fire events which are listened to between various views.

It also employs internal handlers, listening to changes on it’s own attributes to calculate data API URIs for the map and detail Models, efficiently loading in their data via an AJAX call. This prevents duplicate data loading overheads, when the data Models are representing the same URIs.

mapModel & detailModel (data Models)

The mapModel is designed to store data specific to the map, which is often requested from a different data API location to the remaining Graphs and Tables. Therefore, the data Models are separated into two distinct parts.

Both data Models are accessible via references stored in the viewModel and can be passed a dataProcessing function, which performs post processing on the data once loaded, specific to the data and current route.

The detailModel additionally defines a group attribute that keeps track of the current data API grouping (e.g. country, TLD, service type), which affects the Graphs and Table data visualisations.

4.3.3.2 Router

The application has a single Router, which builds on the HTML5 History API. This enables the Backbone History module to change the window’s current URI location without reloading the page. Conversely, the Router listens to these changes and fires internal events within the Backbone application. If the client’s browser doesn’t support the recent HTML5 History API, Backbone will fall-back to using ‘hash-bang’ notation, which appends a /#! after the domain name, forcing the browser to treat it as an anchor link, without sending an HTTP request.

All links in the document must be captured in order to be ‘navigated’ via the History module, described in Listing 4.10. This is done using a jQuery “click” event, listening to all <a> elements that don’t contain a data-bypass attribute. The element’s href
attribute is then checked to make sure it is pointing within the current domain which is then navigated to via the History module.

```
// capture click events on <a> elements without a data-bypass attribute
$(document).on("click", "a:not([data-bypass])", function(_event) {
    var href = { prop: $(this).prop("href"), attr: $(this).attr("href")};
    var root = location.protocol + "/" + location.host;

    // check link is pointing within current domain and protocol
    if (href.prop && href.prop.slice(0, root.length) === root) {
        // prevent sending a new HTTP request to change page
        _event.preventDefault();
        // add to the History API
        Backbone.history.navigate(href.attr, true);
    }
});
```

**Listing 4.10:** Capturing links for the History API

The Router object includes the same JSON web-app route list as the Node server, to map each possible route to its corresponding function label.

### 4.3.3.3 Views

**mainView**

The mainView is the first instantiated view of the Application and contains the core View definitions for each page. It acts as a central controller, listening to route events and calling the corresponding display functions internally. Each page is defined by a ‘route’ function label within the mainView. These functions set the viewModel attributes to represent the current page and define the dataProcessing functions, passed to the data Models.

Each route updates the currently displayed data representations via a setViews() function (Listing 4.11). The function accepts a new list of Views and cleverly performs a ‘set difference’ (using the underscore library) against the old list of views. This ensures that common View such as the mapView and controllerView do not get continually added and removed to the DOM if they persist between pages.

```
setViews: function(_viewList) {
    // calculate the list of new views
    var newViews = _.difference(_viewList, this.viewList) || [];
    // remove the set of old views
    this.removeViews(_.difference(this.viewList, _viewList));
    // add each new view
    for(var i = 0; i < newViews.length; i++){
        var view = newViews[i];
    }
```

**Listing 4.11:** Add new views to the mainView
Listing 4.11: mainView setViews() function

sideView & menuView
These views are instantiated as persistent children of mainView and represent the
sidebar and web-app menu elements respectively. sideView listens to a large range
of attributes on the viewModel and calculates title and description strings for all
headings under the sidebar (Figure B.2). menuView updates the currently active
web-app menu button during transition between pages (Figure B.1).

controllerView
The controllerView is the overriding container for the sliderView and settingsView
and can be added or removed from the content area by the mainView (Figure B.3).
It also supports its own (small) controllerModel which defines what combination
of options it displays to the user for the given page.

Its primary purpose is to provide the user with a central UI ‘controller’ for selection
the currently shown date, current data grouping and advanced filtering settings.

sliderView
The sliderView is users’ main control over the displayed data on the page, enabling
them to intuitively move across a range of dates (Figure B.4). The view listens for
changes in the mapModel data attribute and extracts a list of dates with which to
display on the slider. When users drag the slider element, the View sets the date
attribute back on the viewModel, firing a change in corresponding display objects
e.g. the mapView or tableView.

settingsView
Similar to the sliderView, settingsView allows users to change the current data
filtering method with more advanced queries (Figure B.5). It presents options to
filter data points; with a minimum number of hosts, from specific service types
and from specific TLDs. This view implements jQuery Tree to build the dynamic
checkbox tree for user selection.

searchView
The domain search utility is implemented inside the searchView class using Twitter’s
typeahead library (Figure B.6). Typeahead makes queries to
/search/domain-:domain as the user types in the search field, returning the 20
closest results. On selection of a domain, detailed data for the domain is loaded
in and visualised on further graphs and tables.
mapView

The mapView is present on most pages as a unique element and represents data from the mapModel and current date of the viewModel. It is the most highly inter-coupled class within the application as it must smoothly transition between spherical and flat projections while maintaining a high level of interaction in both states e.g. selecting and zooming in on countries. The map also listens to window “resize” events in order to scale correctly within the dimensions of the window.

panelView

This class is a generic container for tableView and graphView children. It wraps a child View in a set of DOM elements that define size, title text and container styling. Additionally, the panelView listens to Model changes to update linked text in the panel’s title e.g. date or country.

tableView & graphView

Finally, the tableView and graphView classes use the detailModel data to visualise a more detailed representation of the data (Figure B.9 and B.10). There are many types of graphView classes that are defined for specific pages and ways in which to visualise the data. Each one incorporates bespoke listeners for various Model properties and selection types.

All child View objects implement a remove() function that is carefully written to prevent any potential memory leaks, during the use of the web-app. This can happen when event listeners and timing / animation events are not completely cleared when an object is discarded, continually increasing the memory footprint of the application. The best way to prevent this is to define all event listeners via Backbone’s Event module or via a Backbone listenTo() call on Models, which are automatically discarded. However, other library listeners are implemented in the code (such as d3 listeners) which must be handled manually.

4.3.4 Interface implementation

CSS  A significant area of production was the design and coding of the website’s CSS. Similarly to the Jade templates, this is broken into two separate files. style.css defines styling for the core website elements, present on all pages, while index.css defines styles for the elements and visualisations in the web-app. index.css is around 1000 lines long, slowly growing with every new interface feature. Without CSS, many areas of the website will not function correctly for end users, however, web crawlers like search engine indexers will still be able to map the contents of the site correctly.

A range of CSS3 technologies are used to provide enhanced styling possibilities. These include; CSS background-image gradients, custom fonts from Google Fonts, custom symbols generated by fontello.com (Figure 4.1), drop-shadows and text shadows.
Element styles are defined in the CSS files as much as possible, rather than specified in-line or via JavaScript. Therefore, changing a button’s style or country’s highlighting is done by switching the element’s class, instead of setting the style properties explicitly. This decouples styling from technical implementation and enables faster and more cohesive development.

Other technologies like SASS can be used to write CSS descriptions in a more efficient way, similar to using Jade to write HTML. Node also supports a SASS middleware library, for preprocessing and caching SASS files to CSS. However, implementing the middleware with Node proved problematic, so it was avoided to spare development time. [20]

**Text templates** To define the DOM structure for each JavaScript View class in Backbone, `<script type="text/template">` tags are used. These specify usual DOM markup in the page’s HTML, though not rendered by default, they are still referable via an `id` attribute. This creates a useful way of linking complex DOM structures with Backbone Views, without needing to hard-code an unwieldy string. The View classes can then be duplicated any number of times and added anywhere back in the DOM.

Each text template is modularised into its own Jade file and individually imported into the `index.jade` to break up the code. Modifications are therefore easy to make directly into a Jade file, without digging through JavaScript logic. Listing 4.12 shows an example text template that is reused for many `panelView` objects at a time. It defines the DOM structure and also includes `class` attributes to apply the important CSS styling rules.

```html
<script id="template-panel" type="text/template">
  <div class="container">
    <div class="heading">
      <span class="title"></span>
      <div class="icon-help"></div>
    </div>
    <div class="panel"></div>
  </div>
</script>
```

**Listing 4.12:** Example text template script in HTML

A further benefit of using text templates is that they are defined once in the HTML DOM and are available as soon as the page has loaded, without requiring external dependancies.
Underscore templates  Extending the idea of text templates are Underscore’s templates. These compile text string into functions that can be evaluated for rendering. This concept can be combined with text templates by incorporating underscore template markup in HTML, making for a powerful tool with which to render dynamic View objects. Underscore templates can both output variables with `<%= .. %>` and execute JavaScript code using `<% .. %>`.

The web-app makes wide use of underscore templates, most significantly in the panelView for generating visualisation titles and the sideView for dynamic headings and descriptions. Listing [4.13] shows a basic example of the title generation for a panelView object. Underscore first compiles the template function, which can then be called with a set of parameters passed as an object to render the output.

```javascript
// compile a template function
var compiled = _.template("IPv6 Host percentage by <%= group %> for <%= date %>");

// evaluate template with dynamic object
var output = compiled({ group: "Country", date: "March 2013" });

// output -> IPv6 Host percentage by Country for March 2013
```

Listing 4.13: Example underscore template string in JavaScript

4.3.5 Data visualisations

4.3.5.1 Map

All map functionality is contained in the one mapView class. It makes use of much of the d3 library in order to render the map as an SVG to the DOM. Data for the world map is from Natural Earth Data scaled at 1:110m (1 cm = 1100 km) and is supplied as JSON file.[15] To read the file, the topoJSON library is used (Section 4.3.2) to interpret the cultural country boundary arc data into separate paths that can then be rendered to the DOM as SVG `path` elements by d3 (Listing [4.14]).

```javascript
// read JSON file
var world = require('text!static/world.json');

// parse JSON file to JavaScript object
var topology = JSON.parse(world);

var countries = this.svg.append("g")
   .selectAll("path")
   .data(topojson.feature(this.topology, this.topology.objects.countries).features)
   .enter()
   .append("path")
```

Listing 4.14: Example process of reading and rendering the world map
From the outputted `countries` DOM object, d3 can render it using the `d3.geo` projections.\[12\] In this case we are using the two most popular projections `d3.geo.orthographic()` for the globe and `d3.geo.mercator()` for the flat map projection. The projection objects define orientation, scale and translation which are then passed to the `countries` object as a "d" attribute which projects it accordingly, within the bounds of the SVG.

```
// get the bounds of the selected country
var bounds = this.path.bounds(country);

// get the centre coordinates of the country
var translation = [
  -(bounds[1][0] + bounds[0][0]) / 2,
  -(bounds[1][1] + bounds[0][1]) / 2
];

// calculate the scale based on the country bounds and SVG dimensions
var scale = Math.min(0.8* Math.min(this.$svg.width() / (b[1][0] - b[0][0]), this.$spacer.height() / (bounds[1][1] - bounds[0][1])), 15);

// apply the transformation to world SVG elements
countries.attr("transform",
  "translate(" + w + "," + h + ")
  scale(" + scale + ")" +
  "translate(" + translation[0] + "," + translation[1] + ")");
```

Listing 4.15: Calculating the SVG scale for a selected country

To calculate how the map scales when a country is selected the following code in Listing 4.15 is implemented. However, both operations must be able to handle concurrent events being fired, so a global `getCountriesAnimation()` is used to return either a static `countries` object or its `transition` variable. This avoids duplicate animations cancelling one another out, leaving the map in a potentially incorrect state.

Finally, `mapView` uses several event listeners to handle mouse clicks, dragging, window resizing and data updates. Listing 4.16 shows a code overview of setting individual countries colours based on the `mapModel` input data.

```
// get the currently displayed date of the data
var data = this.model.mapModel.get('data');
var date = data[this.model.get('date')];

// calculate the maximum percentage value in the data
var maxPercentage = 0;
for(var i in data){
  for(var j in data[i]){  
    if(maxPercentage < data[i][j].percentage)
      maxPercentage = data[i][j].percentage;
  }
  
  // set the colour for the selected country
  countries[i].attr("fill",
    "hsl(0, 100%, " + maxPercentage + ", 0.6")
  );
}
```

Listing 4.16: Setting individual countries colours based on `mapModel` input data.
// create the d3 colour scale based on maximum value
var scaleDomain = [-1, 0, 0.1*maxPercentage, maxPercentage];
var colourScale = d3.scale.linear()
  .range(["#000", "#C0C19", "#2885B7", "#FFFFFF"])
  .domain(scaleDomain);
// linearly animate the country colours
getCountriesAnimation(500, 'linear')
// set the "fill" colour for each country
.attr("fill", function(d,i){
  // get the Alpha2 country code (used in data API) mapping to ISO code
  // (used by topoJSON)
  var code = Utils.isoToCode(d.id);
  if(date[code]){ // if a code is found, set the corresponding colour
    d.data = date[code];
    return colourScale(d.data.percentage);
  }
  // otherwise return a "no data" colour
  return colourScale(-1);
});

Listing 4.16: Calculating colours for each country element based on mapModel data

4.3.5.2 Graphs

All graphs are implemented in various Backbone Views, however they were initially created as separate but similar Views with minor differences to display the datasets required for each page. This resulted in a large amount of code duplication across each graph and so the implementation was altered to use more generic base Views for a particular graph type with the specific requirements included by extending it. These base views are StackedAreaChartView and LineChartView. The new views extending these were in several cases made to be generic across similar pages to again reduce the level of code duplication. For example PGraphView and PSummaryGraphView are used on the Ping and Path pages and perform a small amount of processing to determine which data to use to calculate points on the graph based on which page it was being used in, as can be seen in Listing 4.17.

initialize: function(_options){
  if (_options.page === 'ping') {
    this.column = 'faster';
    this.columns = ['faster', 'ping6'];
  } else {
    this.column = 'fewer';
    this.columns = ['fewer', 'hops6'];
  }
Functions defined in the base views can be overridden, for example the `dataUpdate` function, which processes data from the model and passes it to the graph. Other small details like chart style and axis labels can also be set by overriding the `initialize` or `dataUpdate` functions, calling the parent view’s function and then setting chart options as shown in Listing 4.18. The `getSeries` function is defined in both the base views and never overridden. This function returns a list of the series to display based on which Country or TLD or Service Type has the most hosts at the most recent date, all other data is then grouped into a Rest of World series when the data is grouped by Country. This is done because there would otherwise be far too many series to display with such small numbers of hosts that the series would not be visible on the graph at all. The function is still used when the data is grouped by Service Type, regardless of the fact that there are always the same four series to display on every graph. This avoids the extra processing required to determine how the data is grouped as well as allowing for new Service Types to be introduced in the future without issue.

```
initialize: function () {
    dualstackGraphView.__super__.initialize.apply(this);
    this.chart.style('expand').yAxis.tickFormat(d3.format(',.2%')).
        axisLabel('Host Distribution');
}
```

Listing 4.18: The parent initialize function is called before setting the chart style, axis formatting and labelling

New functions are also defined in views that extend either `StackedAreaChartView` or `LineChartView`. An example is `ButtonGraphView` which extends `StackedAreaChartView` and includes some jQuery templating and text formatting functions to add buttons above the graph used to change which data is shown on the graph. The additions required for this are shown in 4.19. This is useful when a single graph is being used to show the same comparison for different datasets, for example the graph showing IPv4 and IPv6 reachability broken down by protocol.

```
...templateBottom: $('#template-graph-bottom').html(),
templateButtons: _.template($('#template-data-buttons').html()),

events: {
    "click #data-buttons td": "clickData"
}
```
Every graph needs to respond to changes to the data in the model, so an event listener on the data property is created in the base views. This was also combined with having NVD3 listen to window resize events and call its own internal `chart.update` function, however the fairly complex DOM structure of the website means that sometimes only parts of the web page are resized and so the graph would not scale correctly. This was fixed by using a resize function defined in the base views that sets the height of the SVG proportionally to the new width of the SVG when it is resized as seen in Listing 4.20.

The complex structure also caused another issue with the tooltip displayed on a graph when the mouse is moved over the graph that gives more detailed information about the data. Because only part of the page scrolls rather than the entire page tooltips would not show up if a second graph was present on the page. This is due to tooltips being placed by absolute position by NVD3 and only adjusting the calculation based on how far the entire page had been scrolled. Since only a section of the page had been scrolled NVD3 would place the tooltip off the visible area of the web page, this was fixed by altering the NVD3 library itself to take the relative scrolling of any divs the graph was placed inside into account.

```javascript
resize: function() {
    this.$svg.height(this.$svg.width()*0.5);
},
```

**Listing 4.20:** Resize function used to correctly adjust graph sizes when web page is scaled
4.3.6 Deployment

The final deployment of the web-app rests with the RequireJS configuration. It consists of three parts; the HTML script tag for loading the application, the initial web-app configuration file and the r.js build configuration file.

**Script tag** To load the web-app into the page HTML only one script tag is needed which works seamlessly between development and deployed version of the application. It loads the require.js source file and passes it a reference to the application configuration file via the data-main attribute, Listing 4.21. RequireJS reads this attribute and automatically handles the loading of the remaining files, if there are any.

1 `<script data-main="/js/visual-app" src="/js/lib/require.js"></script>`

**Listing 4.21: RequireJS script tag**

**Web-app configuration** The next step is to define the root file for the web-app, which contains a require.config(); function call. Good practice dictates that all external library files should be defined in the configuration, while application specific files are located relative from the baseUrl. The project’s web-app contains a total of 13 libraries, many of which have inter-dependencies with each other. RequireJS may load libraries out of order, simply as they are called within the JavaScript code, which can cause missing dependency issues. For this reason, ‘shims’ are configured to declare these dependencies to RequireJS, forcing the order in which files are loaded.

Listing 4.22 demonstrates a ‘shim config’ for the underscore string library, which depends on underscore to work. Both libraries also specify a unique namespace with the exports parameter.

1 `require.config({
  baseUrl: '/js/visual',
  paths: {
    'underscore': '../lib/underscore',
    'underscore.string': '../lib/underscore.string'
  },
  shim: {
    'underscore': {
      exports: '_',
    },
    'underscore.string': {
      deps: ['underscore'],
      exports: '_._str'
    }
  }
})`
Build configuration  The last stage is to define the build configuration file, explaining how the web-app should be minimised and the corresponding input and output locations. This configuration file is then run with \texttt{r.js} and Node.js using the following command: \texttt{node r.js -o build.js}.

Because the project’s web-app already has an encapsulating configuration file, it can be used to specify the entire web-app module in the \texttt{build.js} file. Other options include enabling CSS optimisation and removing already condensed files. The final build configuration file can be seen in Listing \ref{lst:build}.

4.4 Deployment & packaging

Extensive modification of the crawler was defined as out of scope for this project, eventually no changes to the crawler were made simply due to extent of the new data processing and complexity of the IPv6 Matrix website. Due to this and outstanding crawler issues, formal packaged version of the complete solution was not created. Instead, the new data processing, data API and IPv6 Matrix website were deployed onto elephant.ecs.soton.ac.uk and the process thoroughly documented. Additional bash and \texttt{init.d} scripts were added for ease of use.
A detailed set of deployment instructions was created, which includes installation of prerequisites and settings configuration.

**Admin scripts** An init.d script was created for management of the node server process, this allows for simple and standard commands such as `service node_ipv6matrix start` or `service node_ipv6matrix stop`. The script accepts `start`, `stop`, `restart` and `status`. Additional start and stop bash scripts were included. A bash script to build all JavaScript and CSS files into a deployment environment was also included.

**Node** Node runs differently when set to be in a production environment. Caching is enabled, debug output is suppressed. Built, minimised and optimised versions of the JavaScript and CSS are delivered. The data API or website components can easily be individually enabled or disabled by environment flags.

**MySQL** The structure of the MySQL database was exported as SQL and can be used to create a new database. The MySQL configuration is tweaked, increasing the index, table and result caches from 8Mb to 64Mb, providing for much faster API response times for all common requests. This is particularly important as due to the nature of the statistical data, nearly all requests require a full table scan to include every row of data in calculations.

**Unix Users** To follow good security convention, the node web server should run as a low privilege user to minimise damage if it was to be compromised. All data and website files are owned by the user `ipv6`. The node server is always started under the `www-data` user which only has read permissions and no superuser or sudo privileges. This prevents node from binding to port 80 to listen to web requests, to circumvent this without escalating privileges temporarily to superuser the node server listens on port 3000 and an iptables entry forwards incoming port 80 requests to port 3000 internally.

**Reliability** The node module forever is used which allows monitoring of the node server, if the web server crashes for any reason it will be automatically restarted. If the server crashes within 1 second of launch forever will stop attempting to restart it as it is likely a configuration error. Forever will wait 1 second after a crash before restart to allow for files and sockets to become available and prevent spinning. All node server errors are logged to `logs/www`. 
Chapter 5

Testing

5.1 Website memory usage

Monitoring memory usage is an important step in ensuring the stability of any software engineering project. During the development of the web-app, informal periodic tests were run to detect any obvious issues during run-time.

JavaScript employs an automatic memory management system that makes use of a ‘garbage collection’ process. This works by freeing memory to any object or variable that becomes unreachable via a chain of references from the document root. Common areas that can cause problems with memory freeing include: timer events and event listeners. If not explicitly cleared, these assignments will still store references to objects that have been potentially removed elsewhere in the code. The Backbone framework offers solutions to assigning event listeners, such that the removable of a View object will also handle the removal of it’s event listeners. This was ensured to be made use of throughout the project’s code.

To monitor the web-app for signs of ‘memory-leaks’, a simple test was run, which involves switching pages over time to perform the addition and removal of view elements, while recording the overall memory allocations using Google Chrome’s developer tools.\[14\]

The test was first run on the website as-is. As no issues had been detected through the period tests run during development, the expected outcome was to see an overall stable count in DOM Nodes and Event listeners. The counts should initially increase as elements are added and removed and then periodically drop as the garbage collector is run.

Memory Test 1, Figure 5.1 shows the results of this initial test. Discouragingly, as the page is changed over time, DOM Nodes and Event listeners are added but are not successfully freed to return the counts back to their original values. This results in a
constant increase in memory usage during the use of the web-app. A problem therefore exists which is causing a memory-leak. Interestingly, DOM Node counts increase almost linearly with the Event listener count, narrowing down potential causes to View any object that is being rendered from scratch on each page transition.

Figure 5.1: Memory Test 1: Web-app memory allocations (as-is) over time

To investigate this further, the most likely code segments that could cause such an issue were identified and individually removed from the subsequent tests, in order to isolate the memory-leak. The first suspected component was the mapView class, as it defines many event listeners through the d3 library, which must be manually handled outside of the Backbone Events module and is present on almost every page.

Memory Test 2, Figure 5.2, demonstrates the result of removing the mapView during the test. Similar to Memory Test 1 the allocation counts are continually increasing, showing no sign of resetting back to their original values. This concludes that the mapView is not the primary cause of the memory-leak.

Figure 5.2: Memory Test 2: Web-app memory allocations excluding mapView over time

Memory Test 3 was run excluding any nvd3 charts from the test, as they are also present on almost all pages and contain potentially the most complex and inter-coupled code of the project. Figure 5.3 displays the resulting allocations. The graph shows the increase
in allocations during page transitions over time, but then a marked decrease as the garbage collector frees memory. At time 12.60s the garbage collector runs for the last time and returns both the DOM Node and Event listener counts back to the starting values. Therefore, it can be inferred that the nvd3 charts are the central cause of the application’s memory-leak issues.

![Figure 5.3: Memory Test 3: Web-app memory allocations excluding nvd3 charts over time](image)

When implementing the nvd3 graphs in the project it was found that no specific ‘remove’ function was defined in its API. However, it is expected that external libraries dereference event listeners and DOM nodes on removable of the containing DOM object. Upon inspection of the library’s inner-workings, it was found that nvd3 keeps a global list of references to all created charts, without providing any way of removing disused chart. This explains the rapid increase in memory footprint, as nvd3 charts are created and constantly referenced in a global list, preventing the garbage collector from ever destroyed their DOM Nodes and Event listeners. This is an issue that will need to be addressed under Future Work, Section 7.

5.2 Data summary

During development a subset of the data was acquired for local testing. This included several small and large TLDs for all crawl dates and a single .com crawl. The .com data was used for load testing as it contains 3.7 million hosts per table.

For metrics that are similar between the previous and new replacement solutions, values could be quickly compared throughout the development process. For new metrics, data types and filtering, manual checks had to take place. This was a lengthy process involving selecting a group with a small hosts and checking those numbers against those found in the CSV crawler output files.
5.3 Data API

5.3.1 Performance

The performance and reliability of both the Data API and website delivery from the node server was tested using multi-threaded HTTP stress testing and benchmarking utility called siege. The tests were run from a virtual machine inside the ECS to allow for maximum bandwidth whilst still responding to external requests. The test simulates real user load from 40 concurrent users each accessing a random URL once a second. The node server was running on elephant.ecs.soton.ac.uk (the current production web server hardware for IPv6Matrix.org) in production mode, on the latest codebase.

The URL test set for the website is simply the small list of HTML, JavaScript, CSS and image files. This list is small as the set of static files required are built and minimised versions when in production. For the data API, a list of 160 API queries was produced, covering the full functionality of the API and including all requests made by the website by default on each page. A sample of this URL list is included as an appendix A.1.

<table>
<thead>
<tr>
<th></th>
<th>Data API</th>
<th>Website files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time:</td>
<td>59.10 secs</td>
<td>59.26 secs</td>
</tr>
<tr>
<td>Data transferred:</td>
<td>49.67 MB</td>
<td>1450.57 MB</td>
</tr>
<tr>
<td>Average response time:</td>
<td>0.81 secs</td>
<td>0.01 secs</td>
</tr>
<tr>
<td>Transaction rate:</td>
<td>14.79 trans/sec</td>
<td>39.17 trans/sec</td>
</tr>
<tr>
<td>Throughput:</td>
<td>0.84 MB/sec</td>
<td>24.48 MB/sec</td>
</tr>
<tr>
<td>Successful transactions:</td>
<td>874</td>
<td>2321</td>
</tr>
<tr>
<td>Failed transactions:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Longest transaction:</td>
<td>2.07 secs</td>
<td>0.11 secs</td>
</tr>
<tr>
<td>Shortest transaction:</td>
<td>0.00 secs</td>
<td>0.00 secs</td>
</tr>
<tr>
<td>Node CPU load:</td>
<td>104%</td>
<td>24%</td>
</tr>
<tr>
<td>MySQL CPU load:</td>
<td>0.1%</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 5.1: Table showing HTTP load testing results for both the Data API and static website files

The results of the load tests highlight a bottleneck in node for the Data API. This was not expected - it was assumed the highest load would be generated by MySQL actually gathering and calculating the result, however this has been highly optimised and cached as can be seen by the 0.1% average CPU usage of MySQL. This node bottleneck is due to the process of formatting MySQL results into JSON, CSV, HTML or XML has an unexpectedly high overhead and has no server side caching.

Throughput and response times (including worst case) for the static website files is perfectly acceptable, as would be expected for simply delivering static files.
Chapter 6

Evaluation

6.1 Data processing

The complete data conversion and analysis process takes approximately 1 hour to complete when run on a full set of crawl data, around the same amount of time as the previous system. However it achieves much more in this time and fixes many outstanding bugs. All 102 gigabytes of previous data collected has been analysed, resulting in 21.1 million rows of summary data for 22 crawl dates. Previously there were only around 3,000 rows per table (15,000 total) available for the front end website to query. The increase 557,000 rows (excluding domains) is due to the extra time and country dimensions added to all datasets, in addition to extra filtering and grouping options and truly represents the wealth of extra data made available by the new data analysis process.

When processing all historical data, the initial step imports 172,000 CSV log files of raw crawler output totalling 102 gigabytes. This creates in total 5630 (74G) of SQLite databases, with the conversion process taking 25 minutes per crawl. These SQLite databases are then summarised, creating a 1.2 gigabytes of summary data which is inserted into the MySQL database, taking 35 minutes per crawl. Most of this time is due to the domain level summary, which analyses all IPv6 metrics for each of the 1 million domains, without this the summary takes only 3 minutes. The resulting MySQL frontend now delivers data from the following tables:

- **HostData** - 280,000 rows - 13.9MB - Combines hosts, dual stack, ping and path into one table, allowing for improvements in disk usage and caching.
- **Reachability** - 200,000 rows - 5.9MB - This data was not processed or available on the previous solution at all. May be filtered and grouped by date, TLD and service.
• **DomainPenetration** - 77,000 rows - 1.9MB - Now includes WWW, MX, NS and NTP options, previously only listed number of IPv6 domains.

• **Domains** - 20,534,000 rows - 1.2GB (160MB indexes) - Provides all data and metrics over all scan dates for each domain out of the 1 million tested.

The entire data processing codebase has been rebuilt from the ground up, resulting in sweeping improvements over the previous system. Conversion speed and accuracy has been improved, and data is converted into more sensible data types. Summary data has many new metrics and dimensions, mainly data over time. It is efficient and robust mainly thanks to being a core underlying component which the rest of the new IPv6 Matrix system relies on to display accurate data, resulting in extensive testing.

### 6.2 API

We have created a comprehensive Data API which enables people to access analysed data with various levels of detail and filtering. As discussed above, the amount of data available has vastly increased with extra time, country dimensions added to all datasets and the ability to request data at a domain level resolution. Results are available in JSON, CSV, HTML or XML dependant on HTTP request type header or URI override.

The API is well documented and follows a simple and sensible URI scheme. The website closely follows the data URI structure and provides links to the data it is currently using - serving as a working example of how to use the API. The solution could easily be extended by strictly defining XML types for each column / data metric which would allow simple expansion to five start open linked data standards through the delivery of .rdf files.

The one feature that is no longer available is the ability to access and query archive data, as this was not considered essential, especially for the average user. This could be added in the future potentially alongside a method to request bulk data such as an exported version of the complete MySQL database, server load and bandwidth would have to be taken into consideration.

MySQL has been thoroughly optimised with extensive caching giving under average response times of under 5ms, and under 400ms response times for fresh requests (which in practice are very rare) as this was assumed to be the bottleneck. Unfortunately subsequent data formatting into JSON, CSV, HTML or XML had an unexpectedly high overhead and is not cached server side. When performing load tests this resulted in underwhelming performance, with an average response rate of 10 requests per second to data API requests. Client side caching will help reduce these requests as node express will correctly set content caching headers. Adding a server side caching layer
such as express middleware express-view-cache or a caching web server proxy such as varnish, memcached or redis would be a simple and effective solution to this.

6.3 Website

The website evaluation is based on both functionality and usability. No formal survey was carried out, but client feedback has been received for the most recent version of the site. Overall, the client has praised the website as being a step forward for the project and is happy with the range of data visualisations. In all areas it has been noted that the new website far excels the previous prototype in terms of both functionality and accessibility. Some minor website accessibility concerns have been noted which will be evaluated further.

6.3.1 Data representation & accessibility

Can users access all data made available by the API?
Yes, the website’s core design was to represent each resource base of the data API by an individual page in the web-app. All pages present a table displaying data selected by date for a given resource. Tables also include surplus columns with post calculated values, further enhancing the quality of the data represented.

Can users intuitively find and select data?
The menu bar presents a clear navigation starting point for finding the appropriate data. Further breakdown can be found throughout each page and should be clearly identifiable via heading titles for all graphs and tables and the map legend. Finally, the table column titles all have associated tooltips, to help users find the correct data metric to sort by.

Selecting data has been made as intuitive as possible with the centralised controller. The slider paradigm is introduced to users on the home page as it animates, demonstrating its use and drawing users’ attention to it. However, it may not be completely clear as to what visualisations are affected by each setting within the controller. The date slider affects both the map and table on most pages but not the graphs, while the group-by selection affects the graphs and table, but not the map. This is the only draw back from containing all settings together, though tooltips on the controller attempt to explain its usage. The alternative would be to disperse the variety of controls across the visualisations, but this would arguably be more confusing for users.

Can users easily interpret visualised data?
The geolocated data represented by the map is very intuitive and links clearly with the date slider paradigm to make sense of the data.
For more advanced visualisations graphs are needed to represent two or more data metrics over time, in two dimensions. Axis titles, hover tooltips and data keys assist in the interpretation of complex data. In some cases, a tabbed button paradigm is used to enable users to select specific data metrics. Combining all available information should allow users to make sense of any graph on the website.

The most detailed breakdowns are shown by the table. Being able to interactively order individual columns by value further aids the interpretation of the relationships between various data metrics.

**Can users find information about visualised data?**

A large consideration for the design of the website was to provide relevant information about the currently visualised data. This is presented consistently in the sidebar, which acts as a go-to for understanding the page’s displayed data and finding its original source from the data API. To supplement this, titles and descriptive tooltips lend a convenient way to explain each metric in detail.

Though many provisions have been made to explain the data clearly, client feedback has suggested that some descriptions (such as the table column headers) are not as clear as they could be. More time could be assigned to writing the interactive tooltip descriptions, to help keep them as consistent and informative as possible throughout the website.

**Can users filter the data to an appropriate level?**

Yes, all data selection features available through the data API have been made available on the website. Therefore, users should have full control over the selection and filtering of all data represented.

Alongside the data and group-by parameters, the controller offers advanced settings with which to filter the data. This was an important request made by the client, in order to maintain existing functionality of the prototype website. The web-app goes one step further by add a minimum sample filter and providing check box tree grouping for the large list of TLDs filter, making it more easily accessible.

### 6.3.2 Website usability & accessibility

**Is the website aesthetically pleasing?**

The styling scheme must be assessed subjectively, in order to judge its aesthetic merits. However, all individuals that have used the website, including the client, have been impressed by the home page introduction and remaining website visuals.

Conscious effort was made to keep the design as consistent as possible across the website, including pages not part of the main web-app. This has paid off, yielding a professional looking website, that functions to represent a complex set of data.
Buttons and links all respond as expected to user actions, following well defined user interface paradigms e.g. hovering over a button changes its visual styling and updates the cursor to a pointer etc. This again maintains consistency throughout the design and promotes intuitive ease-of-use for all visitors to the website.

The website follows a shades of blue colour scheme, with the graphs using a chromatic scale, in order to clearly separate each data point. The map also follows a similar blue to white colour palette, which attempts to highlight as clearly as possible subtle differences in data while following the main website styling. However, the client has expressed interested in being able to evaluate an alternative range of chromatic colour scales for the map e.g. red-green shading schemes.

**Is the website quick to respond to user input?**

There are several user interface interactions that must be responsive to make navigation fluid. The most processor heavy task is to change the page via the web-app menu or country links, which requires most content area elements to be replaced with new ones.

The response time to change pages takes on average, 200-300ms to transition. This is fast enough to allow transitions to feel responsive, and is slightly faster a traditional HTTP request to receive and render the entire contents of the page again.

The date slider has a near instant response time (less than 5ms) in updating the currently displayed date on the map, creating smooth colour transitions between dates. The table is ‘debounced’ to prevent unnecessary rendering of the data on date changes. This allows the rest of the site to stay responsive while incurring only a 250ms delay in updating its contents. Finally, nvd3 charts and the corresponding tabbed buttons also respond very quickly (less than 30ms) to user input.

**Is the website stable to use over time?**

As previously mentioned, a memory-leak issue has been found with the (non)removal of nvd3 charts. This must be addressed in future work before the website can be classed as a stable build. However, no other major issues have been detected. Pages appear to change consistently and data selections and filters update corresponding views without fail.

**Is the website accessible from a range of browser clients?**

The web-app is dependent on modern JavaScript engines and CSS3 which constrains its accessibility to modern browsers only. These include Google Chrome, Safari, Mozilla Firefox, Opera and Internet Explorer 11.

Older browsers that do not support ECMAScript 5.1 implementations of JavaScript will not work with the web-app, but such browsers are dwindling in use. Previous versions of Internet Explorer are currently also not supported due to d3’s selection
engine. However, this is something that can be investigated further, as d3 supports alternative, backwards compatible selection engines for older browsers.

Mobile browsers do render the website correctly, although it is not optimised for smaller sized screens or touch input. This extra functionality was not specified as a high priority for this stage of the work but could easily be implemented later.

### 6.4 VM Packaging and Deployment

The initial goal for packaging the entire solution was to allow distribution to third parties so that they could run their own IPv6 crawler node. This would add more data points across the world allowing for a more complete vision of the IPv6 internet globally by querying the multiple API endpoints. A secondary goal was to ease transition away from the current ageing server hardware towards a virtual machine image that could be deployed on the ECS virtual server stack.

A complete crawler overhaul was left out of the project scope as the scale and time required would be infeasible, but in future would allow for system that was much simpler to distribute to third parties. Currently a third party would require a detailed understanding of the inner operation details of the complete IPv6 crawler system to be able to operate it on a regular basis. Due to a combination of this and other time restrictions as packaging (by nature) was left until after main development of data API and website, a formal packaged version of the complete solution was not created.

A detailed set of deployment instructions was created, which includes every setting and command potentially needed to deploy the node web server and python data analysis to a new production environment, including any prerequisites such as MySQL. This could be further developed into an automated installation script.

Server resilience and reliability was considered for deployment. The node module forever is used which allows monitoring of the node server, if the web server crashes for any reason it will be automatically restarted. Process thrashing and incorrect configuration is detected and prevented, output and errors are managed and added to the correct log files. When in deployment mode, the node server caches files and API requests wherever possible, and delivers minimised versions of the JavaScript and CSS. The MySQL user only has read permissions on the ipv6matrix database for security. The connection from node to the MySQL server uses a pool, allowing for multiple connection threads and recovery from lost connections. The MySQL configuration is tweaked for maximum caching as data is rarely updated, providing for much faster API response times for all common requests. Combined, these provide for a secure, reliable and responsive front end.
To conclude, the goal of an easy to distribute and operate with minimum knowledge of the underlying system was not achieved as it would require a major overhaul of the crawler. However the new data processing and node API and website is simple and easy to deploy and manage. The deployment process is thoroughly documented and therefore it would be trivial to build a fresh Unix virtual machine running the full system, most likely combining the two separate servers into one VM, as with correct process priority management there is no reason for the separation. In this VM image format, visualisation and deployment onto the ECS ecosystem would be possible. Visualisation removes the direct reliance upon a specific hardware set, reducing the current hardware reliability issues (such as drive failures for example) found and offloading this maintenance burden onto technicians, with minimal downtime.
Chapter 7

Future work

7.1 Known issues & bugs

The following tables detail the known issues and bugs remaining in the Crawler (Table 7.1), Data API (Table 7.2) and Website (Table 7.3) and their priority.

7.2 Extensions

The following tables detail the features and extensions for the Crawler (Table 7.4), Data API (Table 7.5) and Website (Table 7.6) both inside and outside the original scope of the project that could be completed in future work.
### Crawler Issue

<table>
<thead>
<tr>
<th>Crawler Issue</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>When MX record field is empty, check for email on A or AAAA records</td>
<td>High</td>
</tr>
<tr>
<td>IPv6Type table - incorrect prefix categorisation stored</td>
<td>High</td>
</tr>
<tr>
<td>Provide enhanced statistics on processing: e.g. remaining files in the log, estimated time</td>
<td>Low</td>
</tr>
<tr>
<td>Time stamp required in the LOG file at start and end of crawl for each domain (needed for tracking of incidents)</td>
<td>High</td>
</tr>
<tr>
<td>Output file and log structure; log folder and .csv not .log</td>
<td>Medium</td>
</tr>
<tr>
<td>Repeated tests on hosts (particularly nameservers and mail servers)</td>
<td>Medium</td>
</tr>
<tr>
<td>CSV is not quoted or escaped when field contains a comma</td>
<td>High</td>
</tr>
<tr>
<td>Geolocation slower than expected</td>
<td>Medium</td>
</tr>
<tr>
<td>Geoip database not updated</td>
<td>High</td>
</tr>
<tr>
<td>Unnecessary rows of output data</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Table 7.1: Existing Crawler Issues**

### Data API Issue

<table>
<thead>
<tr>
<th>Data API Issue</th>
<th>Solution(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slower than desired response times</td>
<td>Server level HTTP response caching: express middleware express-view-cache or a caching web server proxy such as varnish, memcached or redis</td>
</tr>
<tr>
<td>Improve formatting performance</td>
<td>Data API request caching</td>
</tr>
</tbody>
</table>

**Table 7.2: Existing API Issues**

### Website Issue

<table>
<thead>
<tr>
<th>Website Issue</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory leak in removal of nv.d3 graphs</td>
<td>High</td>
</tr>
<tr>
<td>Service type filtering on reachability</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Table 7.3: Existing Website Issues**

### Crawler Feature

<table>
<thead>
<tr>
<th>Crawler Feature</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically update list of domains from Alexa top 1 million</td>
<td>High</td>
</tr>
<tr>
<td>Remove 'inactive' domains after no DNS response for 3 crawls, could reduce crawl time with fewer ping and path test timeouts on old domains</td>
<td>Medium</td>
</tr>
<tr>
<td>Add 'do not crawl' list</td>
<td>High</td>
</tr>
<tr>
<td>Add ability to geolocate a host by IPv6 address</td>
<td>Medium</td>
</tr>
<tr>
<td>Dynamic thread pool size</td>
<td>High</td>
</tr>
<tr>
<td>Intelligently limit based on CPU load and network request rate</td>
<td>High</td>
</tr>
<tr>
<td>Resume from previous position in crawl after reboot / crash</td>
<td>High</td>
</tr>
<tr>
<td>Email crash reports</td>
<td>High</td>
</tr>
</tbody>
</table>

**Table 7.4: Crawler Extensions**
## Chapter 7 Future work

### Data API Feature

<table>
<thead>
<tr>
<th>Feature</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulk data access (all data returned by API currently must be grouped)</td>
<td>Low</td>
</tr>
<tr>
<td>Access and query archived raw data</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Table 7.5: API Extensions**

### Website Feature

<table>
<thead>
<tr>
<th>Feature</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow selection from multiple Data API endpoints</td>
<td>Medium</td>
</tr>
<tr>
<td>Compare data differences between several API endpoints</td>
<td>Low</td>
</tr>
<tr>
<td>Utilise client cookies to store settings (Service Type and TLD filtering options etc.)</td>
<td>Medium</td>
</tr>
<tr>
<td>Label IPv6 day (June 2012) on graphs</td>
<td>Medium</td>
</tr>
<tr>
<td>Multiple y axes and datasets on single graph, for ping and path ratios</td>
<td>High</td>
</tr>
<tr>
<td>Map tooltips (to help understand the navigation features of the map)</td>
<td>High</td>
</tr>
<tr>
<td>Graph, table and map export (images for graphs and map, excel or other format for table)</td>
<td>High</td>
</tr>
<tr>
<td>Map representation selection, currently only shows a single fixed metric per page</td>
<td>Medium</td>
</tr>
<tr>
<td>Automatically update TLD list based on data available to allow for introduction of new TLDs</td>
<td>High</td>
</tr>
</tbody>
</table>

**Table 7.6: Website Extensions**
Chapter 8

Teamwork, planning & progress

Teamwork played a crucial role in the success of the GDP. Having worked together previously, all team members fell into work quickly with little initial management needed to kickstart the project. This proved a great advantage and the strength of the team can be credited for the positive project outcome.

The team held regular meetings to discuss completed and outstanding work, typically daily. Implementation details were constantly being discussed as work was in progress, often resulting in several members working simultaneously on a problem. The team also performed peer code reviews, improving quality and consistency of the codebases.

The project scope covered many disciplines; from optimised database design and querying to delivering responsive and comprehensive interactive data visualisations. Every member took part in all aspects of the project, with different individuals taking natural leads over certain areas of development. This improved everyones expertise in terms of working as a group, but also technical ability in understanding the Python and Javascript languages. Some advanced language concepts were covered during the development cycle and a focus was made on writing well structured, easy to maintain code.

8.1 Gannt chart

Appendix D.0.3 and D.0.4 include the project’s predicted and actual time management Gantt charts. The main difference between the two is that the project scope changed somewhat during the start of the project.

The first research to take place was looking into Open Linked Data, how it worked and what was involved in implementing it. It was soon realised that this would not be appropriate for delivering bulk statistical data, and would hinder further progress on other areas of the project. This resulted in a shift away from Open Linked Data research such as reviewing extraneous data sources with which to combine with crawler data.
Next, the crawler bug backlog was investigated, but once work was underway for other tasks, it was decided with the client that fixing major bugs in the crawler should be left out of scope. A full overhaul of the crawler has been suggested for a future project.

Finally, creating the distributable package was not completed as it was classed as a low project priority and could not be created due to dependancy on the crawler bug fixes.

Overall, time dedicated to research was over provisioned, although, after an initial research stint in weeks 2 and 3 much was done in tandem with project development. Most testing was performed in short periodic stints, with the bulk of the tests being run near the end of the development time.

Core implementation took a much longer time and the website was the most under-provisioned task due to the sheer number of features and complexity this brought about. However, its development took off to a flying start and a working prototype was displayed for Progress Seminar 1 with an almost complete site showcased during Progress Seminar 2.

Similarly, data analysis and processing took longer than expected as reducing computation time and fixing outstanding bugs in the output data had to be undergone. The conversion of the existing data was left till last, so that any errors in the processing code could be mitigated and fixed early on, without squandering time. This shows a good use of time management, as minor problems in the conversion and analysis process could have had serious implications on other parts of the project.

8.2 Organisation tools

Three organisational tools were used to manage the project codebases and task progress.

**Trello** was used an interactive, Agile software development, Scrum style task board (Figure C.1). It features the ability to create, update and move ‘cards’ (similar to sticky notes) that can be visualised in real time and moved between lists to identify their current progress. Trello also includes advanced features like customisable descriptions, task assignments and checklists per card. Using this tool helped remotely organise the team, as all members could see what tasks were in progress and who was working on them.

**Google Drive** was the teams file share for documents outside the codebase e.g. design files and presentation slideshows. This provided a quick, informal method of sharing information while also featuring basic version control.
Git was the VCS of choice for managing all project code files. It provides sophisticated version control features that are required for advanced software development projects. A central repository was setup on a local server, which team members ‘push’ to and ‘pull’ changes from. A detailed break down of the Git repository statistics can be found in Section 8.2.1.

### 8.2.1 Git statistics

The following Git statistics describe the entire involvement in the project’s development.

- **Activity**: 10 weeks, 72 days, 53 active days (74.65%)  
- **Commits**: 557 - average 187 per author, 10.5 commits per active day.  
- **Commits per week max**: 119, min: 22.  
- **Files**: 1723, 6966 file changes.  
- **Lines of Code**: 392594 (731794 added, 339200 removed)  
- **Lines changed, not including adding or removing files**: 42208 (19050 added, 23158 removed)

Figure 8.1: Commits by author over time

Figure 8.1 highlights the consistency over time with which all team members worked on the project and that every member had a fair share of involvement. The graph also
Chapter 8 Teamwork, planning & progress

<table>
<thead>
<tr>
<th>Author</th>
<th>Commits (%)</th>
<th>+ lines</th>
<th>- lines</th>
<th>Active days</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daniel</td>
<td>208 (37.34%)</td>
<td>595736</td>
<td>292342</td>
<td>46</td>
</tr>
<tr>
<td>Jack</td>
<td>175 (31.42%)</td>
<td>143563</td>
<td>40306</td>
<td>32</td>
</tr>
<tr>
<td>James</td>
<td>172 (30.88%)</td>
<td>30506</td>
<td>14260</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 8.1: Per author git commit statistics

Table 8.1 gives the breakdown of commits per author, with the percentage of commits being very similar. It should be noted that large numbers shown in lines added and removed are skewed by the addition and removable of third party libraries during the course of development.

Time statistics, Figure 8.2, demonstrate that Tuesday held the highest number of commits per day of the week, as last minute work was done towards the Progress Seminars. Lastly, Figure 8.3 shows team members working during all hours of the day (and night).

8.2.2 Code statistics

To generate Lines of Code (LoC) statistics, the command line CLOC utility was used, configured to ignore white-space lines.
Lines by language:

- JavaScript LoC (excluding libraries): 3410
- CSS LoC (excluding libraries): 996
- Python LoC: 559

Top 10 largest files:

1. index.css : 724
2. mainView.js : 548
3. mapView.js : 419
4. data.js : 281
5. data.js : 281
6. style.css : 272
7. utils.py : 150
8. sliderView : 147
9. stackedAreaChartView.js : 118
10. controllerView.js : 117
Chapter 9

Conclusion

Improvements have been made by the project team to the amount and quality of data being made available via the website and data API. The open data API is an entirely new feature of the IPv6Matrix project, making summarised data available for anybody to view and use. The API can be extended in the future to include access to bulk summary and raw Crawler data as was available on the previous iteration of the website. The new website is in general much better than its predecessor, displaying a greater amount of customisable data with the inclusion of historical data. This important aspect was missing from the previous website and its use enables website users to see how IPv6 deployment has changed over time. The website was also designed with the intention of making the data interesting and encouraging users to take note of the information being shown. This has been achieved by creating a visually impressive website and using the IPv6 deployment globe as the landing page, demonstrating both the important historical and geographical aspects of the data.

Parts of the project have not been completed due to the main focus being put on the development and redesign of the website. This aspect of the project has the greatest impact as it is the only element externally shown. Further work can be done to improve and extend the project beyond its state at the time of report submission. In particular there are a number of high priority issues and bugs in the Crawler, Data API and Website that should be fixed to improve stability, responsiveness and compatibility. Some of these known issues were existing before the start of this project and while research into the effects of and solutions to these bugs has been performed, no time has been found to fix them. One significant area of the project where research was performed but no work has been completed is the intention to package the Crawler, API and Website as a Virtual Machine. This would allow for better and more efficient utilisation of the hardware available, as well as creating the potential for others to run crawls and provide a separate source of data. This could be used by the IPv6Matrix website as different data sources and allow for comparison of the view of deployment of IPv6 from different locations in the internet.
The development team has worked very well together for the duration of the project in order to produce the new website. Each member has contributed equally to the project, with tasks being completed or directed by those members with a particular interest or prior experience in that area. The project has also been used as a learning opportunity by all members by assisting with or completing tasks where they have little or no previous experience or knowledge. This has enabled each member to gain experience with new languages and technologies.
Bibliography


Appendix A

Documentation
Project Specification And Plan

Title:
IPv6 Deployment Monitoring and Visualisation

Supervisor:
Dr. Tim Chown

Team Members:
James Cotterell
Jack Tench
Daniel Martinho-Corbishley

Customer:
Olivier Crepin-Leblond (chair, ISOC England)

Project Specification:

Recent initiatives by the Internet Society (ISOC), in particular the World IPv6 Launch Event of June 2012 (see http://www.worldipv6launch.org), have led to a slow but steady growth in IPv6 traffic and IPv6-enabled resources on the Internet.

ISOC England has built a suite of tools to measure IPv6 deployment, known as the IPv6 Matrix (see http://www.ipv6matrix.org). This consists of two servers; a backend “IPv6 Crawler” and a frontend website.

The IPv6 Crawler runs through the DNS at pre-set intervals in order to detect, for example, IPv6 DNS servers and IPv6 compliant Web servers, SMTP mailers, and NTP servers, storing the results in .csv format. It provides an ongoing worldwide set of snapshots on the spread of IPv6 accessible content. This Project aims to catalyse the rate of IPv6 adoption by creating and making available a set of tables and graphs showing the spread of adoption, per domain name. Results can be provided on automatically generated Web pages and displayed, for example, per country code top level domain (ccTLD), per gTLD, per IP address etc. – and the classification of results is user configurable.
The graphs currently produced by the Matrix are rather elementary. “The current Web site falls short of the usability and ergonomics to do justice to the wealth of information which has been collected thus far.”

Therefore, the primary focus of the Group Design Project is to redevelop the current IPv6 Matrix website, to provide ease of access to a comprehensive set of analysed data. The main audience consists of the technical community, with the goal of helping provide data for publication and to help present the case for IPv6 adoption to decision makers. It should also be readily accessible to the general public.

The data will be available through a series of online graphical visualisations to highlight trends in IPv6 adoption with relation to geographical location and time. The data will be displayed as a variety of interactive maps, graphs and charts. Whilst maintaining ease of use, the visualisations should allow for a high level of flexibility and detail, providing the user with a range of data filtering parameters. The exact set of visualisations and parameters will be researched based on client feedback and available data sets. A further feature of the website should be the ability to export the generated graphs.

To enable access to the analysed data, an Open Linked Data API will be implemented, providing first and third party access to the information in a multitude of formats. This entails the conversion from the raw output, gathered by the IPv6 crawler, into useful statistical data.

Secondary tasks will be to package the IPv6 Matrix data crawler into a distributable format. The crawler bug backlog will be reviewed in order to fix major issues before distribution. Operation of the crawler currently requires a high level of manual intervention, automating this further should be investigated.

The final distributable will allow a third party to run the IPv6 Crawler from their own location and provide access to the data via the API. Each remote API can then be used as an alternative data source for visualisations on the website. These will provide a better understanding of IPv6 connectivity around the globe.
# Install mysql

# Add users to mysql

# Create mysql db structure

# Edit mysql caching configs?

# Get latest project files

rsync -rzP cave@66.jtench.com:/home/cave/gdp/python/python/
rsync -rzP cave@66.jtench.com:/home/cave/gdp/node/node/
or
git archive --format=tar --remote=cave@66.jtench.com:git/gdp.git HEAD |
        tar xf -

# Install python mysql

sudo easy_install install MySQL-python

or

sudo pip install MySQL-python

# Edit mysql connection configs

node/routes/data.js
python/generateSummary.py
python/completeRegen.py

# Install nodejs

sudo apt-get update
sudo apt-get install -y python-software-properties python g++ make
sudo add-apt-repository -y ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs

# Install requirejs

npm install -g requirejs

# Build require js

node/static/build$ node r.js -o build.js

# Add iptables entry forwarding port 80 to 3000

sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3000

# Start node server

NODE_ENV='deploy' node server.js

# Install forever

sudo npm install forever -g

sudo su - www-data -c "forever stopall"

sudo su - www-data -c "forever list"
su - $user -c "PORT=3000 NODE_ENV='deploy' forever start -l $log_dir/
    forever -o $log_dir/www -e $log_dir/www -a $server_file"

Listing A.1: Deployment instructions
Appendix B

Website components

Figure B.1: menuView DOM element

Figure B.2: sideView DOM element
Appendix B Website components

Figure B.3: controllerView DOM element

Figure B.4: sliderView DOM element

Figure B.5: settingsView DOM element
Figure B.6: searchView DOM element
Figure B.7: mapView DOM element - Orthographic projection

Figure B.8: mapView DOM element - Mercator projection
Figure B.9: graphView DOM element

Figure B.10: tableView DOM element

Figure B.11: Advance options checkbox tree built using jQuery Tree
Appendix C

Screenshots

**Figure C.1:** Trello Scrum style task board
Figure C.2: Advanced settings
Figure C.3: Country Page
Figure C.4: Domain Page
Figure C.5: Dual Stack Page
Figure C.6: Splash Page
Figure C.7: IPv6 Hosts
Figure C.8: IPv6 Hosts Continued
Figure C.9: Ping Data
Figure C.10: Reachability
Figure C.11: Domain Search Typeahead
Figure C.12: Domain Search Data
Appendix D

Gantt charts
### Initial*tasks

- Review*crawler*output
- Research*crawler*operation
- Review*current*website*operation
- Open*linked*data*API
- Review*current*SQL*DB*structure*
- Design*new*/*modified*SQL*DB*structure*
- Research*Open*Linked*Data*
- Design*URI*structure*
- Implement*raw*.csv*conversion*to*SQL*DB
- Implement*data*analysis*
- Test*data*conversion*and*analysis*
- Run*conversion*and*analysis*of*existing*data*
- Implement*URI*mappings
- Test*Open*Linked*Data*API*output
- Website*/*Visualisations
  - Review*existing*website*visualisations*
  - Review*alternative*visualisations*
  - Review*geographical*/*chronological*visualisations*
  - Review*extraneous*data*sources*
  - Pick*&*prioritise*visualisation*set
  - Define*visualisation*parameters*&*filters
  - Research*tools*for*data*presentation
  - Design*site*structure*
- Research*website*implementation
  - Implement*prototype*site*structure
  - Implement*high*priority*visualisations*
  - Finalise*website*implementation*
- Usability*testing*
  - Testing*data*representation*accuracy*
- Crawler*&*API*packaging
  - Review*crawler*bug*backlog
  - Research*packaging*&*deployment*techniques*
- Fix*critical*bugs*
  - Create*crawler*&*API*packages*
  - Test*local*and*remote*packages*
- Project*management*
  - Progress*Seminar*1*prepation
  - Progress*Seminar*2*preparation*
  - Final*report*creation*

---

### Gantt chart

<table>
<thead>
<tr>
<th>Task</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week 5</th>
<th>Week 6</th>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
<th>Week 10</th>
<th>Week 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial<em>tasks</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>crawler</em>output</td>
<td>$\text{11/02/21}$</td>
<td>$\text{11/02/21}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research<em>crawler</em>operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>current</em>website*operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Open<em>linked</em>data*API</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>current</em>SQL<em>DB</em>structure*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design<em>new</em>/<em>modified</em>SQL<em>DB</em>structure*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research<em>Open</em>Linked<em>Data</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design<em>URI</em>structure*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement<em>raw</em>.csv<em>conversion</em>to<em>SQL</em>DB*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement<em>data</em>analysis*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test<em>data</em>conversion<em>and</em>analysis*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run<em>conversion</em>and<em>analysis</em>of<em>existing</em>data*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement<em>URI</em>mappings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test<em>Open</em>Linked<em>Data</em>API*output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Website*/*Visualisations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>existing</em>website<em>visualisations</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>alternative</em>visualisations*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>geographical</em>/<em>chronological</em>visualisations*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>extraneous</em>data<em>sources</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pick*&amp;<em>prioritise</em>visualisation*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Define<em>visualisation</em>parameters*&amp;<em>filters</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research<em>tools</em>for<em>data</em>presentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design<em>site</em>structure*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research<em>website</em>implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement<em>prototype</em>site<em>structure</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement<em>high</em>priority<em>visualisations</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalise<em>website</em>implementation*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability<em>testing</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing<em>data</em>representation*accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crawler*&amp;<em>API</em>packaging</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Review<em>crawler</em>bug*backlog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research<em>packaging</em>&amp;<em>deployment</em>techniques*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fix<em>critical</em>bugs*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create<em>crawler</em>&amp;<em>API</em>packages*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test<em>local</em>and<em>remote</em>packages*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project<em>management</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Progress<em>Seminar</em>1<em>prepation</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Progress<em>Seminar</em>2<em>preparation</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final<em>report</em>creation*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Week 11

- 30/09/2013
- 07/10/2013
- 14/10/2013
- 21/10/2013
- 28/10/2013

### Week 12

- 04/11/2013
- 11/11/2013
- 18/11/2013
- 25/11/2013
- 02/12/2013
- 09/12/2013
Appendix E

Code Snippets

```sql
-- phpMyAdmin SQL Dump
-- version 3.4.5 deb1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Nov 24, 2013 at 02:13 PM
-- Server version: 5.1.69
-- PHP Version: 5.3.6-13ubuntu3.10

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

/* !40101 SET @OLD_CHARACTER_SET_CLIENT = @@CHARACTER_SET_CLIENT */;
/* !40101 SET @OLD_CHARACTER_SET_RESULTS = @@CHARACTER_SET_RESULTS */;
/* !40101 SET @OLD_COLLATION_CONNECTION = @@COLLATION_CONNECTION */;
/* !40101 SET NAMES utf8 */;

-- Database: 'ipv6matrix'

-- --------------------------------------------------------
-- Table structure for table 'DomainPenetration'
-- --------------------------------------------------------

CREATE TABLE IF NOT EXISTS `DomainPenetration` (  
'date' date NOT NULL,
'country' varchar(2) NOT NULL,
'tld' varchar(6) NOT NULL,
'domains' int(11) NOT NULL DEFAULT '0',
'domains6' int(11) NOT NULL DEFAULT '0',
'www' int(11) NOT NULL DEFAULT '0',
```
CREATE TABLE IF NOT EXISTS `Domains` (
    `date` date NOT NULL,
    `domain` varchar(128) NOT NULL,
    `country` varchar(2) NOT NULL DEFAULT 'ZZ',
    `www_hosts` int(11) NOT NULL DEFAULT '0',
    `www_hosts6` int(11) NOT NULL DEFAULT '0',
    `mx_hosts` int(11) NOT NULL DEFAULT '0',
    `mx_hosts6` int(11) NOT NULL DEFAULT '0',
    `ns_hosts` int(11) NOT NULL DEFAULT '0',
    `ns_hosts6` int(11) NOT NULL DEFAULT '0',
    `ntp_hosts` int(11) NOT NULL DEFAULT '0',
    `ntp_hosts6` int(11) NOT NULL DEFAULT '0',
    `http4` int(11) NOT NULL DEFAULT '0',
    `http6` int(11) NOT NULL DEFAULT '0',
    `https4` int(11) NOT NULL DEFAULT '0',
    `https6` int(11) NOT NULL DEFAULT '0',
    `smtp4` int(11) NOT NULL DEFAULT '0',
    `smtp6` int(11) NOT NULL DEFAULT '0',
    `faster` int(11) NOT NULL DEFAULT '0',
    `pingratio` float NOT NULL DEFAULT '0',
    `fewer` int(11) NOT NULL DEFAULT '0',
    `hopratio` float NOT NULL DEFAULT '0',
    KEY `domainsfull` (`domain`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

CREATE TABLE IF NOT EXISTS `HostData` (
    `date` date NOT NULL,
    `country` char(2) NOT NULL DEFAULT '',
    `tld` varchar(10) NOT NULL,
    `type` char(3) NOT NULL,
    `hosts` int(11) NOT NULL DEFAULT '0',
    `hosts4` int(11) NOT NULL DEFAULT '0',
    `hosts6` int(11) NOT NULL DEFAULT '0',
    `dualstack` int(11) NOT NULL DEFAULT '0',
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
CREATE TABLE IF NOT EXISTS `Reachability`
(
  `date` date NOT NULL,
  `country` varchar(2) NOT NULL,
  `tld` varchar(6) NOT NULL,
  `service` varchar(5) NOT NULL,
  `hosts4` int(11) NOT NULL,
  `hosts6` int(11) NOT NULL,
  `reach4` int(11) NOT NULL,
  `reach6` int(11) NOT NULL
) ENGINE = MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- Table structure for table `Reachability`

--

/* !40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/* !40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/* !40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

---

Listing E.1: SQL to build new MySQL ipv6matrix database structure
Listing E.2: Sample of Data API URLs for HTTP load testing
Appendix F

Database structures and contents

F.1 Obsolete SQLite summary database

Table F.1: Structure of table domainPenetration_summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tld</td>
<td>varchar(6)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>total_num_domains</td>
<td>decimal(7,1)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>ipv6_enabled_domains_count</td>
<td>decimal(7,1)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tld</th>
<th>total_num_domains</th>
<th>ipv6_enabled_domains_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>re</td>
<td>12.0</td>
<td>4.0</td>
</tr>
<tr>
<td>lb</td>
<td>54.0</td>
<td>11.0</td>
</tr>
<tr>
<td>ar</td>
<td>2495.0</td>
<td>445.0</td>
</tr>
<tr>
<td>at</td>
<td>3321.0</td>
<td>1209.0</td>
</tr>
<tr>
<td>bw</td>
<td>14.0</td>
<td>3.0</td>
</tr>
<tr>
<td>uz</td>
<td>106.0</td>
<td>14.0</td>
</tr>
<tr>
<td>md</td>
<td>131.0</td>
<td>42.0</td>
</tr>
<tr>
<td>de</td>
<td>44726.0</td>
<td>22114.0</td>
</tr>
<tr>
<td>id</td>
<td>1200.0</td>
<td>381.0</td>
</tr>
<tr>
<td>gb</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>bo</td>
<td>62.0</td>
<td>10.0</td>
</tr>
<tr>
<td>eg</td>
<td>174.0</td>
<td>19.0</td>
</tr>
<tr>
<td>bd</td>
<td>165.0</td>
<td>37.0</td>
</tr>
<tr>
<td>io</td>
<td>42.0</td>
<td>20.0</td>
</tr>
<tr>
<td>hm</td>
<td>9.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>
### Table F.2: Sample content of table domainPenetration_summary (continued)

<table>
<thead>
<tr>
<th>Tld</th>
<th>total_num_domains</th>
<th>ipv6_enabled_domains_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>jm</td>
<td>11.0</td>
<td>6.0</td>
</tr>
<tr>
<td>cm</td>
<td>55.0</td>
<td>9.0</td>
</tr>
<tr>
<td>sb</td>
<td>2.0</td>
<td>1.0</td>
</tr>
<tr>
<td>dm</td>
<td>3.0</td>
<td>2.0</td>
</tr>
<tr>
<td>sg</td>
<td>630.0</td>
<td>287.0</td>
</tr>
</tbody>
</table>

### Table F.3: Structure of table geoip_summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>tld</td>
<td>varchar(6)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>varchar(3)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>varchar(2)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>hosts</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>ipv6hosts</td>
<td>int(5)</td>
<td>Yes</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

### Table F.4: Sample content of table geoip_summary

<table>
<thead>
<tr>
<th>tld</th>
<th>type</th>
<th>country</th>
<th>hosts</th>
<th>ipv6hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>re</td>
<td>NS</td>
<td>FR</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>re</td>
<td>NS</td>
<td>US</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>lb</td>
<td>MX</td>
<td>LB</td>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>lb</td>
<td>MX</td>
<td>US</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>CA</td>
<td>28</td>
<td>6</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>GB</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>JP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>US</td>
<td>39</td>
<td>8</td>
</tr>
<tr>
<td>lb</td>
<td>WWW</td>
<td>LB</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>lb</td>
<td>WWW</td>
<td>US</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>AR</td>
<td>1773</td>
<td>7</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>EU</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>GB</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>US</td>
<td>2054</td>
<td>1268</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>AR</td>
<td>2387</td>
<td>35</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>BR</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>CA</td>
<td>72</td>
<td>4</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>CH</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>DE</td>
<td>114</td>
<td>9</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>EU</td>
<td>15</td>
<td>9</td>
</tr>
</tbody>
</table>
### Table F.5: Structure of table IpDuality_summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>tld</td>
<td>varchar(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>type</td>
<td>varchar(3)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>domains</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>hosts</td>
<td>int(7)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>ipv4</td>
<td>int(7)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>ipv6</td>
<td>int(3)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>ipv4_6</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>no_ip</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

### Table F.6: Sample content of table IpDuality_summary

<table>
<thead>
<tr>
<th>tld</th>
<th>type</th>
<th>domains</th>
<th>hosts</th>
<th>ipv4</th>
<th>ipv6</th>
<th>ipv4_6</th>
<th>no_ip</th>
</tr>
</thead>
<tbody>
<tr>
<td>re</td>
<td>MX</td>
<td>12</td>
<td>15</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>re</td>
<td>NS</td>
<td>12</td>
<td>30</td>
<td>19</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>re</td>
<td>NTP</td>
<td>12</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>re</td>
<td>WWW</td>
<td>12</td>
<td>32</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lb</td>
<td>MX</td>
<td>54</td>
<td>121</td>
<td>104</td>
<td>0</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>54</td>
<td>151</td>
<td>133</td>
<td>0</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>lb</td>
<td>NTP</td>
<td>54</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lb</td>
<td>WWW</td>
<td>54</td>
<td>61</td>
<td>56</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>2495</td>
<td>4726</td>
<td>3000</td>
<td>0</td>
<td>1281</td>
<td>445</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>2495</td>
<td>6007</td>
<td>5293</td>
<td>0</td>
<td>448</td>
<td>266</td>
</tr>
<tr>
<td>ar</td>
<td>NTP</td>
<td>2495</td>
<td>938</td>
<td>912</td>
<td>0</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>ar</td>
<td>WWW</td>
<td>2495</td>
<td>4156</td>
<td>4042</td>
<td>7</td>
<td>107</td>
<td>0</td>
</tr>
<tr>
<td>at</td>
<td>MX</td>
<td>3321</td>
<td>5332</td>
<td>4229</td>
<td>0</td>
<td>773</td>
<td>330</td>
</tr>
<tr>
<td>at</td>
<td>NS</td>
<td>3321</td>
<td>8354</td>
<td>6438</td>
<td>1</td>
<td>1845</td>
<td>70</td>
</tr>
<tr>
<td>at</td>
<td>NTP</td>
<td>3321</td>
<td>2339</td>
<td>2155</td>
<td>0</td>
<td>184</td>
<td>0</td>
</tr>
<tr>
<td>at</td>
<td>WWW</td>
<td>3321</td>
<td>7814</td>
<td>7261</td>
<td>15</td>
<td>538</td>
<td>0</td>
</tr>
<tr>
<td>bw</td>
<td>MX</td>
<td>14</td>
<td>26</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>bw</td>
<td>NS</td>
<td>14</td>
<td>32</td>
<td>23</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>bw</td>
<td>NTP</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bw</td>
<td>WWW</td>
<td>14</td>
<td>11</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table F.7: Structure of table path_summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>tld</td>
<td>varchar(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>type</td>
<td>varchar(3)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Table F.7: Structure of table path_summary (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>hosts</td>
<td>int(5)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>lesshops</td>
<td>int(5)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>hops6</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>hops4</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Table F.8: Sample content of table path_summary

<table>
<thead>
<tr>
<th>tld</th>
<th>type</th>
<th>hosts</th>
<th>lesshops</th>
<th>hops6</th>
<th>hops4</th>
</tr>
</thead>
<tbody>
<tr>
<td>re</td>
<td>MX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>re</td>
<td>NS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>re</td>
<td>NTP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>re</td>
<td>WWW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lb</td>
<td>MX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>6</td>
<td>1</td>
<td>67</td>
<td>62</td>
</tr>
<tr>
<td>lb</td>
<td>NTP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lb</td>
<td>WWW</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>29</td>
<td>0</td>
<td>354</td>
<td>325</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>78</td>
<td>43</td>
<td>719</td>
<td>789</td>
</tr>
<tr>
<td>ar</td>
<td>NTP</td>
<td>4</td>
<td>0</td>
<td>44</td>
<td>36</td>
</tr>
<tr>
<td>ar</td>
<td>WWW</td>
<td>10</td>
<td>3</td>
<td>110</td>
<td>107</td>
</tr>
<tr>
<td>at</td>
<td>MX</td>
<td>43</td>
<td>2</td>
<td>578</td>
<td>403</td>
</tr>
<tr>
<td>at</td>
<td>NS</td>
<td>603</td>
<td>107</td>
<td>7493</td>
<td>6789</td>
</tr>
<tr>
<td>at</td>
<td>NTP</td>
<td>88</td>
<td>8</td>
<td>967</td>
<td>799</td>
</tr>
<tr>
<td>at</td>
<td>WWW</td>
<td>245</td>
<td>13</td>
<td>2709</td>
<td>2169</td>
</tr>
<tr>
<td>bw</td>
<td>MX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bw</td>
<td>NS</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bw</td>
<td>NTP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bw</td>
<td>WWW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table F.9: Structure of table ping_summary

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>tld</td>
<td>varchar(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>type</td>
<td>varchar(3)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>hosts</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>faster</td>
<td>int(6)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>delay6</td>
<td>decimal(19,3)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>delay4</td>
<td>decimal(19,3)</td>
<td>Yes</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
### Appendix F Database structures and contents

#### Table F.10: Sample content of table ping_summary

<table>
<thead>
<tr>
<th>tld</th>
<th>type</th>
<th>hosts</th>
<th>faster</th>
<th>delay6</th>
<th>delay4</th>
</tr>
</thead>
<tbody>
<tr>
<td>re</td>
<td>MX</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>re</td>
<td>NS</td>
<td>11</td>
<td>0</td>
<td>654.278</td>
<td>131.341</td>
</tr>
<tr>
<td>re</td>
<td>NTP</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>re</td>
<td>WWW</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>lb</td>
<td>MX</td>
<td>13</td>
<td>3</td>
<td>388.664</td>
<td>359.496</td>
</tr>
<tr>
<td>lb</td>
<td>NS</td>
<td>16</td>
<td>7</td>
<td>1246.143</td>
<td>1462.154</td>
</tr>
<tr>
<td>lb</td>
<td>NTP</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>lb</td>
<td>WWW</td>
<td>2</td>
<td>2</td>
<td>115.597</td>
<td>116.501</td>
</tr>
<tr>
<td>ar</td>
<td>MX</td>
<td>1274</td>
<td>307</td>
<td>46816.157</td>
<td>44878.220</td>
</tr>
<tr>
<td>ar</td>
<td>NS</td>
<td>196</td>
<td>93</td>
<td>13786.825</td>
<td>13375.257</td>
</tr>
<tr>
<td>ar</td>
<td>NTP</td>
<td>26</td>
<td>22</td>
<td>252.427</td>
<td>292.741</td>
</tr>
<tr>
<td>ar</td>
<td>WWW</td>
<td>101</td>
<td>79</td>
<td>1555.575</td>
<td>2460.268</td>
</tr>
<tr>
<td>at</td>
<td>MX</td>
<td>745</td>
<td>149</td>
<td>25823.545</td>
<td>21995.931</td>
</tr>
<tr>
<td>at</td>
<td>NS</td>
<td>1607</td>
<td>352</td>
<td>50130.579</td>
<td>43407.437</td>
</tr>
<tr>
<td>at</td>
<td>NTP</td>
<td>178</td>
<td>10</td>
<td>5822.763</td>
<td>4168.816</td>
</tr>
<tr>
<td>at</td>
<td>WWW</td>
<td>514</td>
<td>54</td>
<td>16383.633</td>
<td>12156.669</td>
</tr>
<tr>
<td>bw</td>
<td>MX</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>bw</td>
<td>NS</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>bw</td>
<td>NTP</td>
<td>0</td>
<td>0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>bw</td>
<td>WWW</td>
<td>1</td>
<td>1</td>
<td>0.709</td>
<td>2.784</td>
</tr>
</tbody>
</table>

#### F.2 MySQL summary database

##### Table F.11: Structure of MySQL table HostData

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>date</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>char(2)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tld</td>
<td>varchar(10)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>char(3)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>hosts4</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>hosts6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>dualstack</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>noip</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pinghosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>faster</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### Table F.11: Structure of MySQL table HostData (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping4</td>
<td>float</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ping6</td>
<td>float</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pathhosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fewer</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>hops4</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>hops6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Table F.12: Sample content of MySQL table HostData

<table>
<thead>
<tr>
<th>date</th>
<th>country</th>
<th>tld</th>
<th>type</th>
<th>hosts</th>
<th>hosts4</th>
<th>hosts6</th>
<th>dualstack</th>
<th>noip</th>
<th>pinghosts</th>
<th>faster</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-09-13</td>
<td>CN</td>
<td>ac</td>
<td>WWW</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>DE</td>
<td>ac</td>
<td>WWW</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>GB</td>
<td>ac</td>
<td>WWW</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>IN</td>
<td>ac</td>
<td>WWW</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>JP</td>
<td>ac</td>
<td>WWW</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>TR</td>
<td>ac</td>
<td>WWW</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>US</td>
<td>ac</td>
<td>WWW</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>VN</td>
<td>ac</td>
<td>WWW</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>ZZ</td>
<td>ac</td>
<td>NS</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>AT</td>
<td>ac</td>
<td>NS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>CN</td>
<td>ac</td>
<td>NS</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>DE</td>
<td>ac</td>
<td>NS</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>GB</td>
<td>ac</td>
<td>NS</td>
<td>12</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>IN</td>
<td>ac</td>
<td>NS</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>JP</td>
<td>ac</td>
<td>NS</td>
<td>39</td>
<td>39</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>NL</td>
<td>ac</td>
<td>NS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>TR</td>
<td>ac</td>
<td>NS</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>US</td>
<td>ac</td>
<td>NS</td>
<td>36</td>
<td>36</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>VN</td>
<td>ac</td>
<td>NS</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-09-13</td>
<td>ZZ</td>
<td>ac</td>
<td>MX</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table F.13: Structure of MySQL table DomainPenetration

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>date</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>varchar(2)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tld</td>
<td>varchar(6)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>domains</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
### Table F.13: Structure of MySQL table DomainPenetration (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>domains6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>www</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mx</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ns</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ntp</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Table F.14: Sample content of MySQL table DomainPenetration

<table>
<thead>
<tr>
<th>date</th>
<th>country</th>
<th>tld</th>
<th>domains</th>
<th>domains6</th>
<th>www</th>
<th>mx</th>
<th>ns</th>
<th>ntp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-08-21</td>
<td>US</td>
<td>ac</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>CN</td>
<td>ac</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>TR</td>
<td>ac</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>DE</td>
<td>ac</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>JP</td>
<td>ac</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>VN</td>
<td>ac</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>GB</td>
<td>ac</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>ZZ</td>
<td>ac</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>IN</td>
<td>ac</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>FR</td>
<td>ae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>DE</td>
<td>ae</td>
<td>19</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>HK</td>
<td>ae</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>RU</td>
<td>ae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>NL</td>
<td>ae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>NZ</td>
<td>ae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>PK</td>
<td>ae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>CA</td>
<td>ae</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>CH</td>
<td>ae</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>AE</td>
<td>ae</td>
<td>140</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>EG</td>
<td>ae</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table F.15: Structure of MySQL table Reachability

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>date</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>varchar(2)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tld</td>
<td>varchar(6)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>service</td>
<td>varchar(5)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hosts4</td>
<td>int(11)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table F.15: Structure of MySQL table Reachability (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>hosts6</td>
<td>int(11)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reach4</td>
<td>int(11)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reach6</td>
<td>int(11)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table F.16: Sample content of MySQL table Reachability

<table>
<thead>
<tr>
<th>date</th>
<th>country</th>
<th>tld</th>
<th>service</th>
<th>hosts4</th>
<th>hosts6</th>
<th>reach4</th>
<th>reach6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-08-21</td>
<td>DE</td>
<td>ac</td>
<td>SMTPS</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>DK</td>
<td>ac</td>
<td>SMTPS</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>GB</td>
<td>ac</td>
<td>SMTPS</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>JP</td>
<td>ac</td>
<td>SMTPS</td>
<td>24</td>
<td>0</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>NL</td>
<td>ac</td>
<td>SMTPS</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>TR</td>
<td>ac</td>
<td>SMTPS</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>US</td>
<td>ac</td>
<td>SMTPS</td>
<td>17</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>VN</td>
<td>ac</td>
<td>SMTPS</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>CN</td>
<td>ac</td>
<td>HTTPS</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>DE</td>
<td>ac</td>
<td>HTTPS</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>GB</td>
<td>ac</td>
<td>HTTPS</td>
<td>11</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>IN</td>
<td>ac</td>
<td>HTTPS</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>JP</td>
<td>ac</td>
<td>HTTPS</td>
<td>19</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>TR</td>
<td>ac</td>
<td>HTTPS</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>US</td>
<td>ac</td>
<td>HTTPS</td>
<td>25</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>VN</td>
<td>ac</td>
<td>HTTPS</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>CN</td>
<td>ac</td>
<td>HTTP</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>DE</td>
<td>ac</td>
<td>HTTP</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>GB</td>
<td>ac</td>
<td>HTTP</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>IN</td>
<td>ac</td>
<td>HTTP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table F.17: Structure of MySQL table Domains

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>date</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>domain</td>
<td>varchar(128)</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>varchar(2)</td>
<td>No</td>
<td>ZZ</td>
<td></td>
</tr>
<tr>
<td>www_hosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>www_hosts6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mx_hosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>mx_hosts6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Table F.17: Structure of MySQL table Domains (continued)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null</th>
<th>Default</th>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns_hosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ns_hosts6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ntp_hosts</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ntp_hosts6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>http4</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>http6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>https4</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>https6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>smtp4</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>smtp6</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>faster</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>pingratio</td>
<td>float</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>fewer</td>
<td>int(11)</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>hopratio</td>
<td>float</td>
<td>No</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table F.18: Sample content of MySQL table Domains

<table>
<thead>
<tr>
<th>date</th>
<th>domain</th>
<th>country</th>
<th>www_hosts</th>
<th>www_hosts6</th>
<th>mx_hosts</th>
<th>mx_hosts6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-08-21</td>
<td>acmt.ac</td>
<td>US</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>acs.ac</td>
<td>US</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>alastairc.ac</td>
<td>GB</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>badminton.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>bbt.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>blanche.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>bollywood.ac</td>
<td>US</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>britneyspears.ac</td>
<td>US</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>cic.ac</td>
<td>GB</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>digitaluniversity.ac</td>
<td>IN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>door.ac</td>
<td>ZZ</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>dtm.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>fishing.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>fox.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>fukuchan.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>google.ac</td>
<td>US</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>hello.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>hixhix.ac</td>
<td>VN</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>homoeopathy.ac</td>
<td>JP</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2010-08-21</td>
<td>imax.ac</td>
<td>GB</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>