Teaplot v0.1: A browser-based 3D engine for animating TEACUP experiment data

Isaac True*, Grenville Armitage, Philip Branch
Centre for Advanced Internet Architectures, Technical Report 150828A
Swinburne University of Technology
Melbourne, Australia
itrue@swin.edu.au, garmitage@swin.edu.au, pbranch@swin.edu.au

***Abstract*—TEACUP is a software tool developed at CAIA for running automated TCP experiments, which can generate static PDF graphs of experiment results using R. Teaplot v0.1 is an extension to this tool which allows the user to interactively visualise and animate these results in a web browser in both 2-and 3D.**

This report contains information pertaining to the architecture and technical requirements of the extension, along with the installation procedure and a user interface guide for the web client component.

Ideas for potential further work on Teaplot were identified during development, including bug fixes and solutions for performance and visual issues, and have been included within this report.

I. Introduction

TEACUP [1] is a software tool developed at CAIA for running automated TCP experiments and modelling complex network characteristics and performance in a testbed. Teaplot v0.1 is an extension to this tool.

Traditionally, TEACUP experiment data is analysed and plotted using scripts written in R [2], which generate static graphs as PDFs. Changing how the data is displayed means modifying command line arguments and regenerating these graphs, repeatedly – a time-consuming and repetitive job. Teaplot reinvents this process by providing the user with a graphical interface in their browser which can be used to plot experiment data dynamically, allowing the user to pick and choose results from various experiments, combine multiple measured and calculated metrics, and animate the data as it changes over time across multiple graphs, in both 2- and 3D.

The current TEACUP version as of the time of writing is v1.0. Further information relating to TEACUP, its operation, and example scenarios for testbeds can be found in the TEACUP technical reports [3] [4] [5] [6].

*The work described in this report was done during the author's winter internship at CAIA in 2015

This report is intended to give the reader a technical overview of the functions, requirements, and usage of Teaplot. Section II details how the system is structured, while Section III details the technical requirements and software dependencies. Section IV describes how to install Teaplot using TEACUP v1.0 as a base. Section V guides the potential user in the use of the client web interface component of Teaplot, and Section VI gives examples for potential further work that have been identified as useful or beneficial to the performance of the program.

II. Architecture

A. System Design

The Teaplot system is separated into two components: the client, and the server. The client component is an HTML5-compliant WebGL-based web page which is used as the user interface for Teaplot, while the server component, written in Python, is responsible for the mathematical and data processing tasks required in creating the data for the animations. All intensive mathematical calculations are performed by the server, where the operations can be accelerated through the use of mathematical libraries such as SciPy and NumPy, which provide Python wrappers around native libraries. The server is also responsible for communicating with TEACUP, in order to extract and utilise the data it makes available. This architecture can be seen in Figure 1.

The server makes available to the client an API, through which the client can request a list of experiments, metrics, and flows, and also request a set of data to be generated for a particular experiment, metric, and flow combination in order to plot and animate it on screen. This design ensures separation of the client and server logic, and facilitates the use of the server-side mathematical libraries as described above.

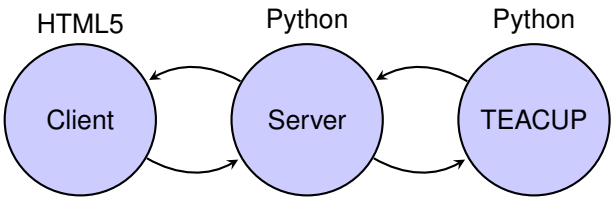

Fig. 1. System Architecture Diagram

The server also provides the HTTP server for the static web pages comprising the client; to use Teaplot, a user simply navigates to the URL of the server using a compatible web browser.

The Teaplot server is implemented as a Fabric [7] task called `animate`, and so easily integrates into existing TEACUP installations. The `animate` task is easily executed as an argument to the `fab` task. Details regarding this usage can be found in Section V.

B. API

The Teaplot server API utilises JSON (JavaScript Object Notation) [8] formatted documents for communicating data.

Table I shows a list of endpoints present in the API, which the client application can make use of, and can be used for debugging, if need be.

As the server program is assumed to be short-lived and in a controlled environment, no authentication takes place on the API.

C. Server

The server component is built using Django [9], a Python-based framework for building websites. It was chosen due to its reliability, asynchronous nature, and wide selection of website utility functions, which facilitated the construction of the API.

The data processing functions make heavy use of the NumPy [10] and SciPy [11] libraries, which are Python wrappers for manipulation of matrices and arrays in native high-performance linear algebra libraries written in C and Fortran.

To facilitate ease of integration into the Fabric framework used by TEACUP, the server is run using uWSGI [12], which creates and manages the server using Django's WSGI interface.

D. Client

The client is built using a combination of HTML5 and JavaScript, along with a number of JavaScript libraries. The base UI framework used is Bootstrap 3 [13], supplemented by jQuery UI [14]. The application also makes use of the templating engine `pure.js` in order to greatly simplify construction of tables and lists in HTML.

jQuery [15] is used for a multitude of tasks, but is primarily used for simplification of communicating with the server API using its JSON functions.

TABLE I
API ENDPOINTS

Endpoint	Type	Description
`/api/default`	`GET`	Default parameters for the client (exp_id, metric, etc.)
`/api/metrics`	`GET`	Lists the available metrics, including SIFTR and Web10G metrics if enabled
`/api/metrics/get`	`POST`	Starts the extraction process for the experiment and metric combination sent as POST data and returns the available data
`/api/experiments`	`GET`	Lists the available experiments
`/api/graph`	`POST`	Generates plottable data points for the experiment, metric, and flow combination sent as POST data, and returns them as (x,y,z) coordinates
`/api/paths`	`GET`	Returns a list of the directories in use by Teaplot for debugging purposes **n.b. not JSON formatted**

III. TECHNICAL REQUIREMENTS

A. Server

As the server performs the bulk of the calculations, a computer with a powerful CPU and enough RAM to comfortably perform the required data manipulation is recommended, especially if the server host is a virtual machine. The exact performance requirements will be heavily dependent on the size of the data sets and experiments being used.

The Teaplot server component requires Python 2.7 and the following Python libraries to be installed:

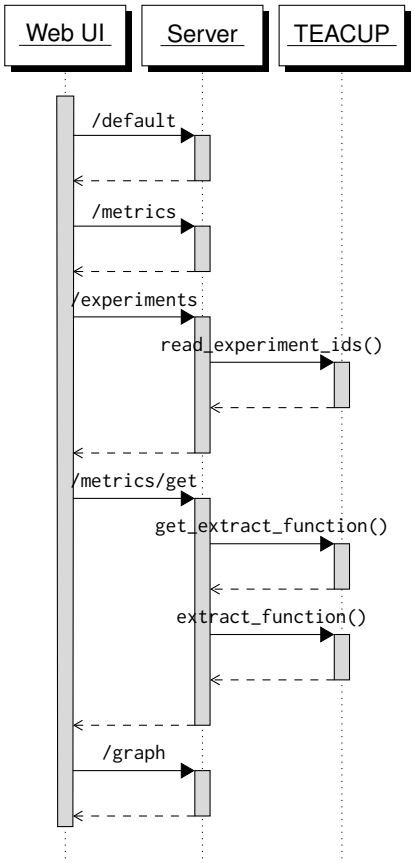

Fig. 2. Teaplot API Request Sequence Diagram

- Django v1.8.0
- NumPy v1.11.4
- SciPy v1.11.4

It also requires the following additional software and appropriate dependencies to be installed:

- uWSGI v2.0.11
- SPP v0.3.6 [16]
- TEACUP v1.0

The following JavaScript libraries are required for the client UI, and are included in a minified form in the installation package:

- bootstrap.js v3.3.5
- jquery.js v2.1.4
- jquery-ui.js v1.11.4
- beebole-pure.js v2.83 [17]
- stats.js d869e3f61c [18]
- require.js v2.1.19 [19]
- three.js r71 [20]

Note: Software package version listed above are the versions that were used during the initial development of the platform. Future and previous versions of packages may or may not work with Teaplot, and could create issues or require code modification.

B. Client

No accompanying software is required to be installed on the client's device; the Teaplot client component takes place entirely within the browser. It requires a modern WebGL compatible version of Google Chrome, Chromium, or Firefox. Microsoft Edge and Internet Explorer have not been tested, but recent versions may be compatible.

As the data is loaded in memory for plotting, large data sets may require a large amount of RAM (i.e. more than 1GB) to be available on the client's device. Furthermore, the graphics card, WebGL implementation, and graphics drivers on the client's computer must support shaders. All modern Intel, Nvidia, and AMD graphics cards support WebGL shaders, along with their Windows-based driver counterparts. However, if the browser is running on a Linux-/Unix-based system, driver-side support may be problematic, especially when using open-source implementations of the drivers. Problems could also arise if the web browser is being run inside a virtual machine host.

Due to the large RAM requirements of the client application, the user may quickly reach the hard 4GB memory address space limit if they are using a 32-bit web browser. As such, it is recommended to use a 64-bit version of the operating system, along with a 64-bit web browser binary. Windows users are recommended to use the 64-bit version of Firefox Nightly [21].

It is recommended that for best compatibility and performance, the web browser is run on a non-virtualised host running a 64-bit version of Linux, Windows, or FreeBSD, along with a 64-bit web browser, using the manufacturer-supplied proprietary/non-free drivers.

IV. Installation Procedure

The following installation procedure outlines the actions needed to install Teaplot on a FreeBSD-based system. Commands prefixed with # denote that the execution should take place inside a shell with root privileges, while $ denotes user-level privileges.

See Appendix A for information regarding installation on FreeBSD 9.x.

1) If TEACUP v1.0 is not already installed, retrieve it from http://downloads.sourceforge.net/project/teacup/teacup-1.0.tar.gz. Additional information can be obtained from the CAIA technical reports [3] [6].

2) Extract Teaplot installation archive into the TEACUP directory(`$TEACUP_DIR`)[1] :
```
$ cd $TEACUP_DIR
$ tar xvf teaplot-0.1.txz
```
3) Apply the Teaplot patch for `fabfile.py` inside the same directory:
```
$ patch -p1 < teaplot.patch
```
4) Install system dependencies. For example, on FreeBSD[2]:
```
# pkg install py27-django py27-scipy py27-numpy uwsgi spp
```
5) Installation is now complete. Teaplot can now be run from the fabfile directory (`$FABFILE_DIR`) [3] as a Fabric task called "animate":
```
$ cd $FABFILE_DIR
$ fab animate
```

V. Usage

A. General

The following assumptions are made when running a TEACUP Fabric task, and thus Teaplot:

- `fab` is being executed in a directory containing a a link to or copy of TEACUP's `fabfile.py`, and a relevant and properly configured `config.py`
- The file `config.py` defines `TPCONF_script_path`, which points to the extracted TEACUP source tree (`$TEACUP_DIR`)
- The Fabfile directory (`$FABFILE_DIR`) is home to one or more experiment directories that contain actual raw experiment results files.

B. Task Arguments

Table II shows the arguments that can be added to the Fabric task to customise the functionality of Teaplot. All arguments are optional, and are supplied to the Fabric task like so:

```
$ fab animate:arg1="value1",arg2="value3",. . .
```

Values should be enclosed by quote marks (i.e. `". . . "`) or apostrophes (i.e. `'. . . '`). A value of `''` generally denotes a null, or blank, value. For the arguments `out_dir` and `exp_dir`, a value of `''` indicates that Teaplot should use the current working directory as that directory.

`exp_list` and `exp_list` are interpreted as relative to the current working directory (the directory in which the Fabric task is executed). `out_dir` is relative to the experiment directories, as each one is processed by TEACUP. `animate_dir` is relative to the directory defined by `TPCONF_script_path`.

Table III shows the additional arguments for the Fabric task that relate to the defaults to use for the web client.

For example, if the user wishes to have the server listen on IP address `0.0.0.0` (all available interfaces) and display three graphs by default, the fabric task would be as follows:

```
$ fab animate:address="0.0.0.0",graph_count="3"
```

TABLE II
Fabric task arguments for Teaplot

Argument	Description
`address`	IP address for the server. Default: `'127.0.0.1'`
`port`	Port number for the server. Default: `'8000'`
`animate_dir`	Teaplot directory containing the Django WSGI application. Default: `animate`
`out_dir`	Directory in which to generate TEACUP intermediate files. Default: `''`
`processes`	Number of uWSGI processes to use. Default: 1
`threads`	Number of uWSGI threads for each process. Default: 1
`exp_list`	Location of the experiments_completed.txt file generated by TEACUP. Default: `'experiments_completed.txt'`

C. Launching the Client

In order to visualise TEACUP experiment data, the following steps should be carried out:

1) Start the web server component using the method outlined in the previous section (ensuring that the IP address and port chosen are reachable by the client)
2) Open the web browser on the client computer, and navigate to the IP address and port pair used by

[1] TEACUP directory refers to the directory in which the TEACUP installation archive was extracted. In the TEACUP `config.py` this is referred to as `TPCONF_script_path`.

[2] A minimum of `spp-0.3.6` is required. At the time of writing, the latest version available in the FreeBSD package repositories is 0.3.5. Thus, manual installation from source could be required. Refer to the SPP website for installation instructions [16].

[3] Fabfile directory refers to a directory separate to TEACUP, which contains a copy of `config.py` and a copy of or link to TEACUP's `fabfile.py`, in which the `fab` command is executed in order to generate experiment data. This directory contains one or more "experiment directories", which contain the experimental results generated by TEACUP.

TABLE III
FABRIC TASK ARGUMENTS FOR DEFAULT CLIENT SETTINGS

Argument	Description
source_filter	TEACUP source filters (semicolon-separated) to use as default in the web client. Default: ''
test_id	TEACUP test IDs (semicolon-separated) to use as default in the web client. Default: ''
metric	Default TEACUP metrics (semicolon-separated). Default: ''
graph_count	Default number of graphs to display. Default: '1'
graph_names	Names of the graphs displayed (semicolon-separated) by default. Default: ''
lnames	Legend/Flow names to show on the graphs (semicolon-separated). Default: ''
siftr	Enable ('1') or disable ('0') SIFTR. Default: '0'
web10g	Enable ('1') or disable ('0') web10g. Default: '0'
etime	Default end time in seconds for a new graph. '0' infers no default end time. Default: '0'
stime	Default start time in seconds. Default: '0'

the server
e.g. http://127.0.0.1:8000

3) If desired, increase the number of graphs to be shown on-screen by clicking the "Graph" button in the top navigation bar, and then modifying the "Number of graphs" control as desired. Alternatively, restart the server with the desired number as the value for the argument graph_count. The name of each graph can also be optionally set here, or using the argument graph_names.

D. Selecting Tests and Flows

1) The Teaplot client web UI should now be visible on-screen. Now, open the metric selection dialogue by clicking "Metrics" in the navigation bar, and select one or more test ID's (which the Teaplot server reports it has found in the experiments_completed.txt file identified by exp_list), and optionally provide source filters in the appropriate text box. For information on the source filter syntax, refer to the TEACUP technical report [4].
2) Select the "Metrics" tab, and select one or more metrics for extraction and processing. If SIFTR or Web10G metrics are required, the server should be restarted with the additional argument siftr='1' or web10g='1'.
3) Click the "Update Flows" button to send the requested parameters to the server. The server will then extract the metric data from the selected experiments, and apply the source filter(s) to the data. The metrics dialogue will close once this operation has successfully completed. If an error has occurred, refer to the file teaplot.log inside $FABFILE_DIR for more information. This process may take tens of seconds, especially if a specific metric and test ID combination has not previously been extracted.
4) Open the "Flow" dialogue by clicking the "Flow" button in the navigation bar. The metrics and accompanying flows [1] available for display will be displayed here. The flows are grouped by metric, and then alphabetically sorted. The flow start times displayed are calculated relative to the earliest flow start time in each group. [2]
5) Select a number of flows to show on the graph using the check boxes. Once a flow is selected, it will be added to the "Y Axis Flow Mapping" table. From here, the graph on which to display the data can be selected. Furthermore, alternative axis configurations can be selected. *However*, as of version 0.1, selecting anything other than "Time" for the X Axis and "Nothing" for the Z Axis is considered experimental; different flow and metric configurations may not work or be compatible.
6) Once the desired flows have been selected, click the "Update View" button to send the request. As may take several seconds, as the server must process the data, normalise time stamps, and transfer the data for the client, where it is then plotted on the appropriate graph. The number of flows and total number of data points will heavily influence how long this process will take. [3]

E. Interactivity

Once the "Update View" process is complete, the dialogue will close and the data will appear on the appropriate graphs. The X (time) axis limits will automatically adjust to include the entire data set; the X axis of each graph is identically scaled, with the same start and end times. The Y and Z (if selected) axes will automatically scale to include the highest value on each graph; these axis are not linked (i.e. each graph has its own independent scale). If `stime` or `etime` arguments were specified with the Fabric task, the graphs will automatically adjust the X axis to the designated values. If desired, the minimum and maximum values of each axis can be adjusted using the sliders in the "Control" panel. These values are represented as percentages of the maximum value on each axis. All graphs are linked and controlled by the same sliders. If no name is currently set for a graph, it will be updated with the metric it is currently displaying.

Individual flows are represented by a single colour, irrespective of metric, in order to ease visual comparisons of the same flow on multiple graphs. If desired, names can be assigned to individual flows by clicking the flow name in the "Legend" panel, or by providing a semicolon-separated list of names via the `lnames` argument when starting Teaplot.

The graph view is controlled using the mouse. In 2D view (the default), panning can be achieved by clicking and holding the right mouse button, while zooming is achieved using the scroll wheel.

F. Button Functions

3D view can be toggled (for experimental 3D plots) using the "Toggle 3D View" button found in the top navigation bar. Clicking the "Reset View" will reset the position of the graph view to the default.

Clicking "Toggle Grid" will show or hide the grey dividing lines shown on the graphs.

The "Animate" button begins the graph animation, the duration of which can be controlled using the "Animation Time" slider found in the "Controls" box. This animation produces a bar which moves left-to-right across all graphs simultaneously, "uncovering" data points as it moves. This can be used to visualise the interaction between metrics over time. Clicking the "Animate" button while the animation is running will pause and resume the animation.

The "Export Command" button displays a dialogue containing a list of arguments which can be passed to the Fabric task the next time the server task is executed. These arguments represent the current state of the web client interface and current selections made by the user, so that the user can record these arguments and restart the server in more-or-less the same state, with the same metrics, for example, at a later time.

VI. Potential Further Work

A. Finalisation of 3D Graphing Functions

As previously outlined, the 3D graph functions of Teaplot are currently considered experimental, as not all combinations of metrics are able to be successfully plotted. The current implementation on the server makes use of the SciPy class `interpolate.InterpolatedUnivariateSpline` to generate an interpolation function using the experimental data, in order to create a 3D mapping against a second data series and time.

Further work in this area could involve identifying combinations of metrics which break the interpolation system, and solving the issues relating to that. Furthermore, the system could be extended to produce 3D scatter plots using three distinct metrics using a bivariate interpolation class, such as `interpolate.BivariateSpline`.

The file relating to the server-side processing of the data is `animate/api/teaplot.py`.

B. User Interface

Many refinements of the user interface can be made. Perhaps chief among the potential refinements is the grouping of table rows in the "Metrics" and "Flows" dialogues into collapsible experiment/test/run groups, in order to tidy up the interface and simplify the selection process when working with large sets of experiments.

A further refinement is the separation and/or grouping of SIFTR [23] and Web10G [24] metrics in the metric selection dialogue. Currently, the list is very cluttered

[1] A "flow" refers to a one-way data stream between an IP address and port pair, shown in the format `<source IP>_<source port>_<destination IP>_<destination port>`.
e.g.: `172.16.10.3_28176_172.16.11.3_8000`

[2] A "group" refers to an experiment/test grouping as calculated by TEACUP; a single group contains a single test run.

[3] The amount of data transferred between the client and server during this process is approximately 1MB per 100,000 data points. This could be greatly reduced (if required) through the use of transparent HTTP compression such as `gzip`, which could be introduced through the use of a reverse-proxy web-server application on the server host, such as `nginx` [22].

when both web10g and siftr flags are enabled; grouping the metrics into collapsible standard/Web10G/SIFTR tables could go a long way to tidying up that interface and making it easier for the user.

Such changes should be made to the css/teaplot.css, js/teaplot.js, and index.html files inside the animate/static/ directory.

Additional changes can be made to the Bootstrap CSS settings in order to further reduce visual clutter and size of lists. Changes should be made on the Bootstrap customisation page at http://getbootstrap.com/customize/ using the /animate/static/config.json file included with Teaplot as the base. The resulting Bootstrap files can safely overwrite the existing files in Teaplot.

C. Grid Rendering

Currently, the grids are not being calculated correctly, and only appear in the proper dimensions using the current statically set dimensions of the graph. The grid rendering code should be replaced (in animate/static/js/graph.js), in order to allow for dynamically resizeable graphs and grid spacing.

D. Memory Usage

Substantial reduction in memory usage is possible if the three.js Geometry objects are changed to BufferGeometry objects, which will move a large chunk of the used browser memory into buffers on the GPU. However, this may take some effort and knowledge, as implementing shaders with BufferGeometry objects is not as simple as with normal Geometry.

Further reductions are possible using point thinning server-side, in order to reduce the amount of data points being rendered simultaneously by the client. However, a considerable bonus is had by rendering all of the available points, as the user is currently able to zoom in to see the high resolution of the points. A potential work-around is to have a "streaming" interface between the client and server, where the client requests the data at a resolution/point thinning factor relative to the zoom level in real-time.

Changes to the 3D rendering in the Teaplot client should be made in the animate/static/js/graph.js file.

E. Text

The axis labels rendered on the graphs seem to be affected by some size limit – long labels are cut off. The cause of this is unknown, however text rendering in three.js/WebGL is known to be problematic. Potential solutions are to move text rendering out of the WebGL canvas, instead creating HTML text labels that float in the correct positions on the graph.

As this is related to 3D rendering in the Teaplot client, changes should be made in the animate/static/js/graph.js file.

Long test IDs currently break some sections of the UI, namely the "Y Axis Flow Mapping" table and the "Legend" panel. The CSS relating to these UI components should be modified to be fixed-width, and to use proper text wrapping.

These modifications can be made inside animate/static/css/teaplot.css.

VII. Conclusion

This report has described the architecture, functions, dependencies, and technical details pertaining to Teaplot v0.1, the browser-based 3D visualisation extension for TEACUP v1.0, along with ideas for potential bug fixes and improvements which were identified during development.

Acknowledgements

We would like to thank Jonathan Kua and Rasool Al-Saadi for their invaluable help with testing Teaplot and providing sample data, and Djuro Mirkovic for providing the original foundation for the 3D graphing engine.

Appendix
FreeBSD 9.x Installation

The easiest method for installing the required packages on FreeBSD is to use the pkgng framework, rather than the standard ports tree. Firstly, pkgng must be configured. This can be done by executing the following commmands:

```
# portsnap fetch extract
# echo "WITH_PKGNG=yes" >> /etc/make.conf
# make -C /usr/ports/ports-mgmt/pkg install clean
# pkg2ng
```

Following this, the package repository needs to be initialised:

```
# mkdir /etc/pkg
```

Edit the contents of /etc/pkg/FreeBSD.conf to be the following:

```
FreeBSD: {
url: "pkg+http://pkg.FreeBSD.org/${ABI}/latest",
mirror_type: "srv",
enabed: "yes"
}
```

Lastly, execute:

```
# pkg update
```

Following the successful execution of these commands, the pkgng framework can be used to install packages.

REFERENCES

[1] TCP Experiment Automation Controlled Using Python (TEACUP). [Online]. Available: http://caia.swin.edu.au/tools/teacup/

[2] The R Project for Statistical Computing. The R Foundation. [Online]. Available: https://www.r-project.org/

[3] S. Zander and G. Armitage, "TEACUP v1.0 - A System for Automated TCP Testbed Experiments," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529A, 29 May 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529A/CAIA-TR-150529A.pdf

[4] S. Zander and G. Armitage, "TEACUP v1.0 - Data Analysis Functions," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529B, 29 May 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529B/CAIA-TR-150529B.pdf

[5] S. Zander, "TEACUP v1.0 - Command Reference," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150529C, 29 May 2015. [Online]. Available: http://caia.swin.edu.au/reports/150529C/CAIA-TR-150529C.pdf

[6] S. Zander and G. Armitage, "CAIA Testbed for TEACUP Experiments Version 2," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 150210C, 10 February 2015. [Online]. Available: http://caia.swin.edu.au/reports/150210C/CAIA-TR-150210C.pdf

[7] Fabric. [Online]. Available: http://www.fabfile.org/

[8] ECMA-404 The JSON Data Interchange Standard. ECMA International. [Online]. Available: http://json.org/

[9] Django Project. Django Software Foundation. [Online]. Available: https://www.djangoproject.com/

[10] NumPy. [Online]. Available: http://www.numpy.org/

[11] SciPy. [Online]. Available: https://www.scipy.org/

[12] uWSGI. [Online]. Available: https://github.com/unbit/uwsgi

[13] Bootstrap. Twitter, Inc. [Online]. Available: http://getbootstrap.com/

[14] jQuery UI. The jQuery Foundation. [Online]. Available: https://jqueryui.com/

[15] jQuery. The jQuery Foundation. [Online]. Available: https://jquery.com/

[16] Synthetic Packet Pairs (SPP) – Tool for passive round trip time measurement. [Online]. Available: http://caia.swin.edu.au/tools/spp/

[17] pure.js. BeeBole. [Online]. Available: http://beebole.com/pure/

[18] stats.js. [Online]. Available: https://github.com/mrdoob/stats.js/

[19] RequireJS. [Online]. Available: http://requirejs.org/

[20] three.js. [Online]. Available: http://threejs.org/

[21] Firefox Nightly. Mozilla Foundation. [Online]. Available: https://nightly.mozilla.org/

[22] nginx. Nginx, Inc. [Online]. Available: http://nginx.org/

[23] siftr. The FreeBSD Project. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=siftr

[24] Web10G. The Web10G Project. [Online]. Available: https://www.web10g.org/