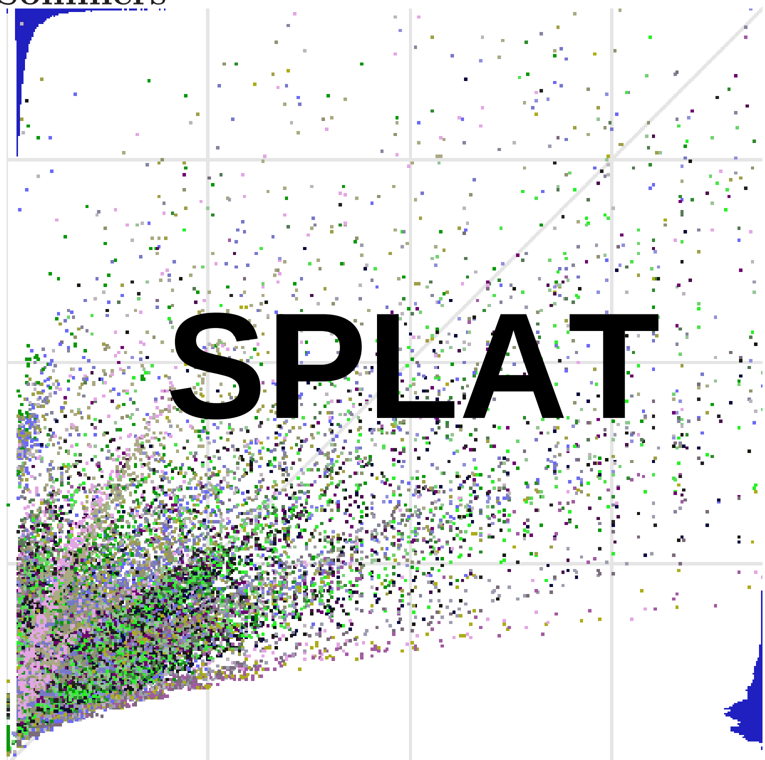
SPLAT

A Scatter and Phase Plot Animation Tool
User Manual

Joel Sommers



This manual is for SPLAT: a scatter and phase plot animation tool. The following copyright notice covers the SPlAT source code, including all documentation, images, and ancillary files.

Table of Contents

1 Overview and Installation of SPLAT

Visualizations provide a natural means for organizing large complex data sets and mining them for characteristics of interest. This manual describes SPLAT, a **s**catter and **p**hase p**l**ot **a**nimation **t**ool. SPLAT offers a broad set of capabilities for investigating Internet measurement data sets based on scatter and phase plots—two well-known techniques for exploratory data analysis. An important feature of SPLAT is that it can animate the scatter and phase plots over time to reveal dynamic characteristics of the data at hand. We demonstrate SPLAT's capabilities through a series of case studies that show how both general profiles and important non-obvious details in large Internet data sets can be identified thereby illustrating its utility for a diverse set of network research areas.

1.1 Basic Capabilities of SPLAT

As the name suggests, SPLAT offers visualization capabilities based on 2D scatter and phase plots of data. SPLAT includes a set of pruning, zooming and feature selection (*e.g.*, filtering) capabilities developed specifically for large high-dimensional Internet data analysis. A distinguishing feature of SPLAT is that it can display animations of phase and scatter plots as they evolve over time. This 3D capability greatly facilitates the discovery and identification of subtle features in the data that may reveal interesting aspects of Internet structure and behavior but would typically be overlooked when using a purely static display of the data. We are not aware of any widely-used visualization tools that have the combination of capabilities provided by SPLAT.

Phase plots and scatter plots are well-known exploratory analysis tools for determining association and examining relationships among different variables. In its basic form, phase plot analysis considers two time-dependent variables $x(t), y(t)$. A phase plot is a graph of all points $x(t_i), y(t_i)$ over a specified period of time where the x-variable is plotted on the horizontal axis and the y-variable on the vertical axis. Since Internet data sets almost always have time components associated with them, they are naturally suited to phase plot analysis. Scatter plots are, in essence, identical to phase plots but do not have an implicit time axis. A time axis can be trivially added, however, to study the temporal evolution of the two variables.

The basic design requirement for SPLAT is to display a phase plot, to allow one to zoom and pan on specific regions, and to view the animations of the plot over time. In addition, \textscsplat has a number of annotation capabilities, including display of relative point densities along each dimension of the plotting region, listing of the current time in the trace file (for animations), coloring of data points to highlight possible associations with higher-layer entities (*e.g.*, packets associated with flows), and labeling of these higher-level entities (*e.g.*, a string representation of the five-tuple that defines a particular flow).

For large, multivariate data sets, a key design requirement is the ability to view subsets of the main phase plot data, conditioned along one or more dimensions. To enable filtering, SPLAT can load auxiliary data sets (*e.g.*, time series data that are synchronized with the main phase plot data) and various kinds of categorical summary data (*e.g.*, estimates of flow round-trip times or flow sizes). For example, assume that our basic phase plot data consists of spacings between individual packets of a flow as the packets *enter* a congested router queue and the spacings of the same packets as they *exit* the queue. We could make use of time series data of the queue length to enable visual detection of correlations between phase plot features and congestion events. Similarly, we may wish to restrict our view of the ingress-egress phase plot to consider the largest flows that also have round-trip times within a certain range, or to view only phase plot data for flows having destination IP addresses matching a given prefix. These built-in capabilities of SPLAT distinguish it from more general-purpose visualization tools such as GGobi that are not designed to handle Internet-specific data sets. Additional concrete examples of some of SPLAT's filtering capabilities are described in the case studies, below.

SPLAT is written in C++ and uses the cross-platform Trolltech Qt librariesSee `http://www.trolltech.com`. for its graphical capabilities. Plotting areas are drawn using OpenGL widgets, enabling relatively simple zoom, translation, and rotation by manipulating the world-to-screen and projection matrices. Saving a plot for later reference is handled by converting the raw frame buffer data to a common image format. All scatter plots in this manual were produced using this capability in SPLAT. SPLAT includes the ability to read a variety of common Internet-related data types.

The rest of the manual describes how to build and use SPLAT. Please also see [Postscript], page 25 for the original technical paper describing SPLAT.

1.2 Installing SPLAT

SPLAT uses the Trolltech Qt libraries, at least version 4. For non-commercial users, open-source versions of Qt can be found at `http://www.trolltech.com/developer/downloads/qt/index`. Before attempting to compile SPLAT, you should download, build, install, and test Qt.

When building Qt, you must ensure that the following libraries are built: QtGui, QtXml, QtOpenGL, and QtSql. Please refer to the Qt documentation for help in properly configuring and building Qt.

Qt is designed to run on most UNIX/X11 platforms, Windows, and MacOS X. Nearly all testing for SPLAT has been performed on MacOS X, however some testing has been performed on Linux. There are a number of known bugs (and likely many more unknown) that have been found on both Mac and Linux platforms and we are grateful for any fixes or advisories you may have as you use SPLAT. SPLAT has never been built on

Windows—proceed at your own risk. (Note: I don't believe there are any endianness issues within the SPLAT code—any problems are likely to be in implementation differences between Qt platforms, especially with respect to OpenGL and the windowing system).

Once you've built and installed Qt, and after you've appropriately modified your PATH variable to include the Qt tools, you can do the following to build SPLAT:

```
./configure
make
```

The configure script locates qmake, does some simple checking of headers and builds a Makefile. You should check whether the output of configure makes sense, in particular that it finds the version of Qt that you want it to find. Many Linux systems have older versions of Qt installed by default (version 3.x) since Qt forms the basis of the KDE. SPLAT won't work with Qt version 3, so please be aware.

Using the Makefile created by configure will cause qmake to do its magic and build the SPLAT binary.

For experts and/or hackers, note that the SPLAT project file is named '`splat.qt`'. It should more-or-less work on basic Mac and X11 systems, but you may wish to tweak it to suit your needs.

2 Basic Use of SPLAT

This chapter gives an overview of how SPLAT works, starting from raw data to viewing the data in SPLAT. Further details on data input and the SPLAT user interface are found in later chapters.

2.1 Creating Input for SPLAT

Making suitable input for SPLAT has been designed to be relatively pain-free. Splat uses XML configuration files to set up the data environment and reads data from white-space delimited plain text files.

Columns of the input text file are associated with an axis (x, y, or z/time) through tags in the configuration file. There are also specific tags to define the data type for the x and y columns/axes, axis labels, data type precision, and data ranges for x, y, and z/time axes.

For example, we may have a very simple input data set consisting of three columns and four rows:

```
1   1   0.0
2   2   1.0
3   3   5.0
4   5   6.0
```

Let's say that we store this data in the file '`input.txt`'. We can then create a configuration file consisting of the following:

```
<splat_data>
    <plot_data filename="input.txt"
               name="configuration file example"
               xcol="0" ycol="1" zcol="2"
               xtype="int"
               xlabel="my label for x axis" xunits="widgets"
               xrange="0:10"
               ytype="int"
               ylabel="my label for y axis" yunits="bars"
               yrange="0:10" />
</splat_data>
```

Based on the above configuration file Splat will read the data in '`input.txt`' (see **filename** tag), treating data in the 0th column as the x-axis data, column 1 as the y-axis data, and column 2 as the z/time axis (see **xcol**, **ycol**, and **zcol** tags in the example above). The x and y data types are defined as integers (see **xtype** and **ytype** tags) and the range of these axes are defined as between 0 and 10, inclusive (see **xrange** and **yrange** tags). A meaningful name has been given to this data set (see **name** tag). Labels have been defined for the x and y axes (see **xlabel** and **ylabel** tags) and unit names have been defined (see **xunits** and **yunits** tags). Note that these last five tags can be any arbitrary string. Also, note that all column numbers are zero-based.

At present, only one **plot_data** element can be defined within the outer **splat_data** tags. It is intended that future versions will be able to accomodate multiple **plot_data** tags. Also, Splat will eventually be able to configured to read data from a network address or from a SQL database.

Note that the input data can be gzip-compressed. The filename above would be '`input.txt.gz`' if the data for that example was compressed.

2.2 Starting SPLAT

Depending on your platform, Splat may be started by double clicking on an icon (*e.g.*, MacOS X) or by starting it in the usual say from a command shell, (*e.g.*, UNIX/X11 platforms).

When splat is initially started, a simple window should appear similar to the one shown in Figure 2.1. From the menu(s) of this window, you can load a configuration file and specify general display preferences.

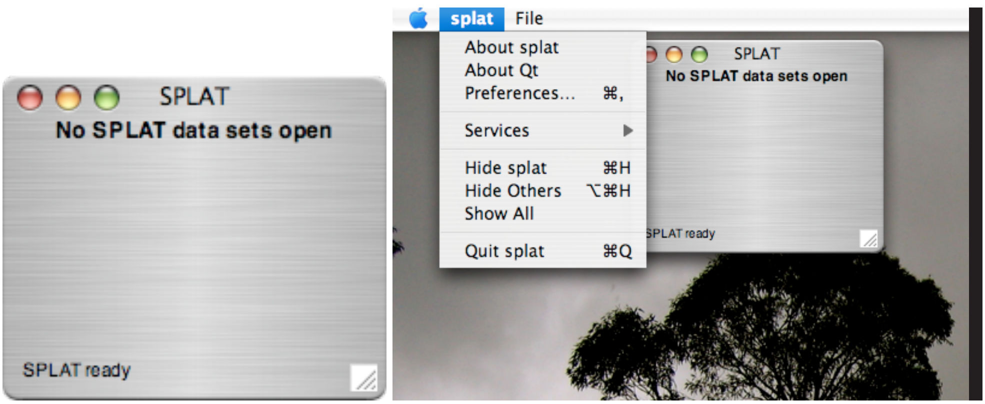


Figure 2.1: Base window and menus for Splat.

As data sets are opened and closed, a list of open data sets is shown in the main Splat window. Windows corresponding to these data sets can be hidden or made visible via the checkboxes. An example of the main Splat window with one data set open is shown in Figure 2.2.

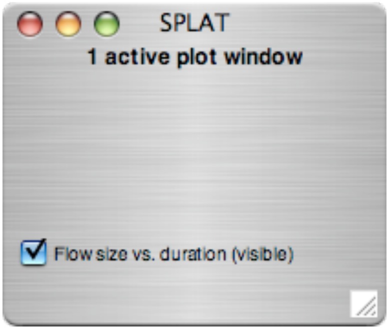


Figure 2.2: Base window for SPLAT once one data set has been opened.

2.3 User Interface Elements

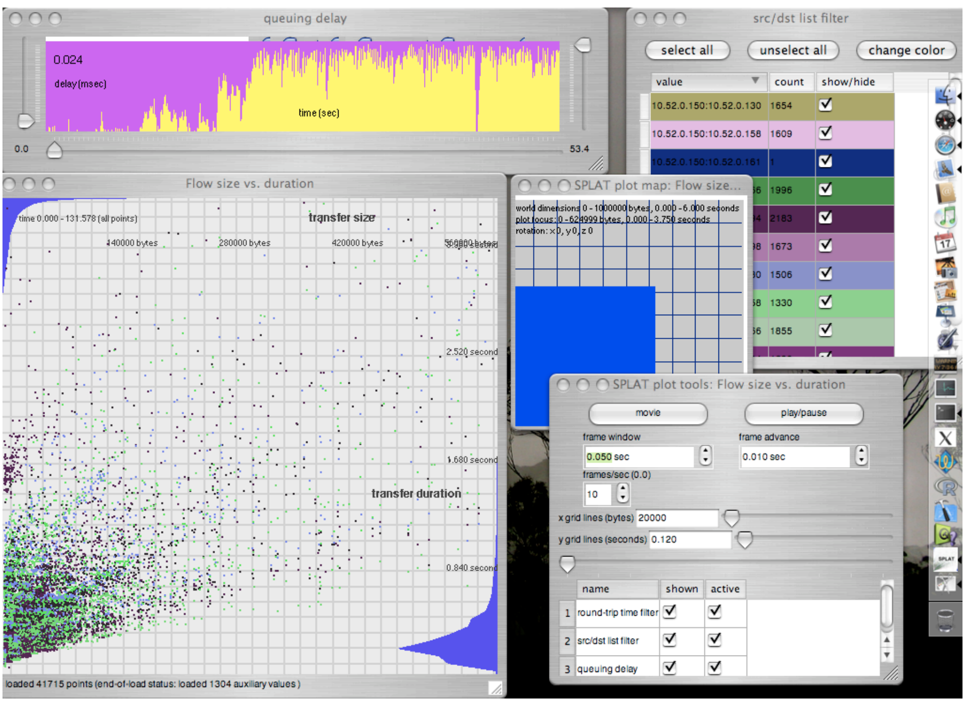


Figure 2.3: Screen shot of SPLAT

The user interface for SPLAT consists of a main plotting area, where the scatter/phase plot data is shown. There is a window to modify certain aspects of the display (plot tools) such as animation settings and grid line spacings. Another window indicates the relative location of the plotting area with respect to the entire configured data space (plot map). These three windows are *always* initially displayed.

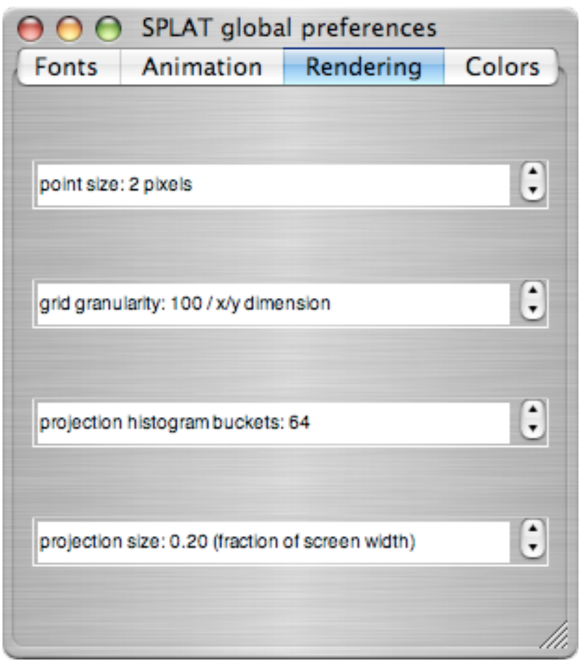
2.3.1 Filter Windows

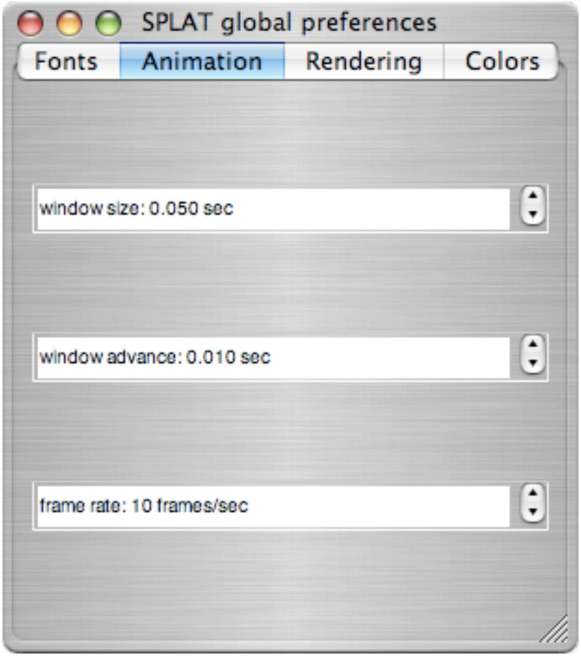
In Figure 2.3 there are two additional windows shown: a time series filter window and a list-style filter window.

3 The SPLAT User Interface

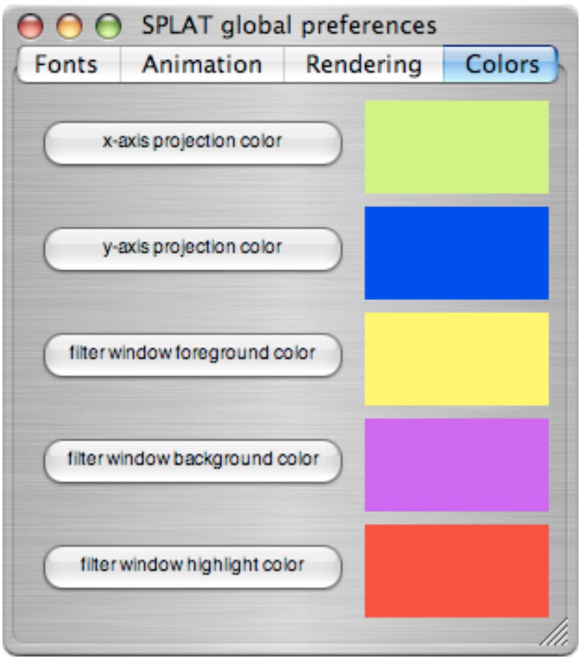
This chapter will provide details on various aspects of the SPLAT user interface. There are some relevant notes below but most of this chapter remains unwritten.

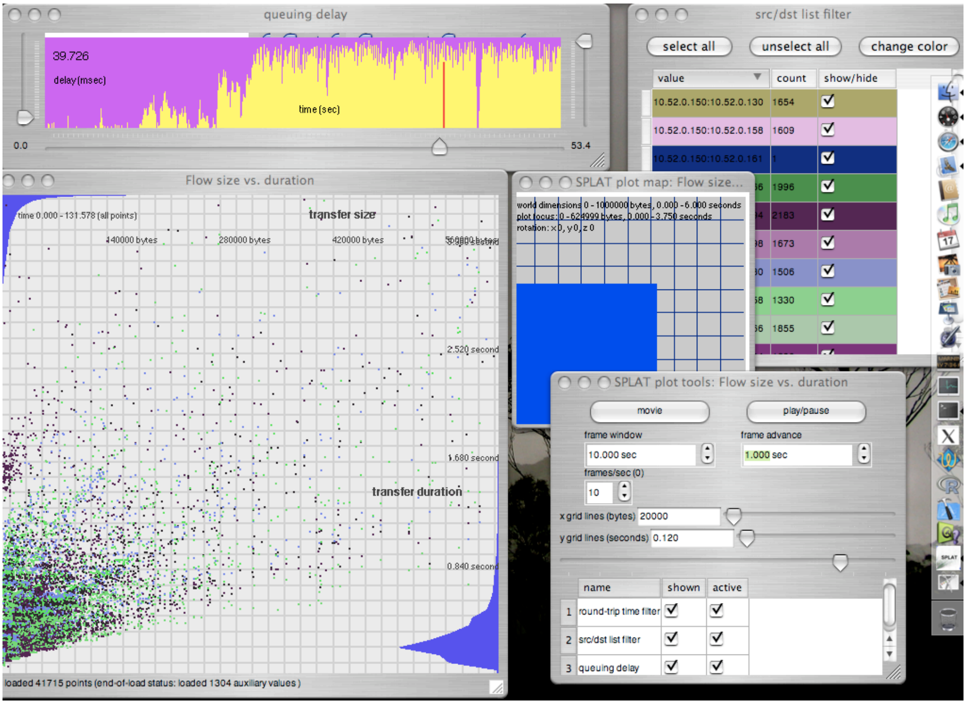
Should note with fonts preference which specific items are affected.

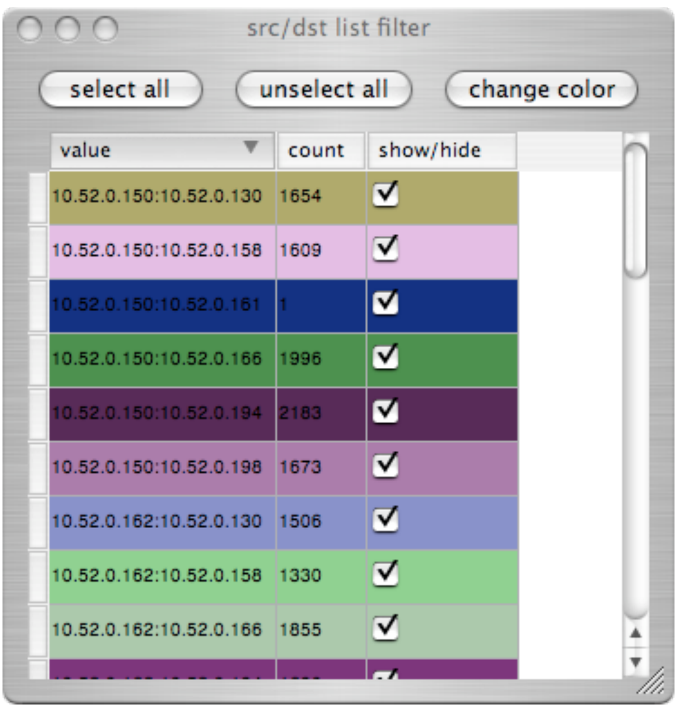


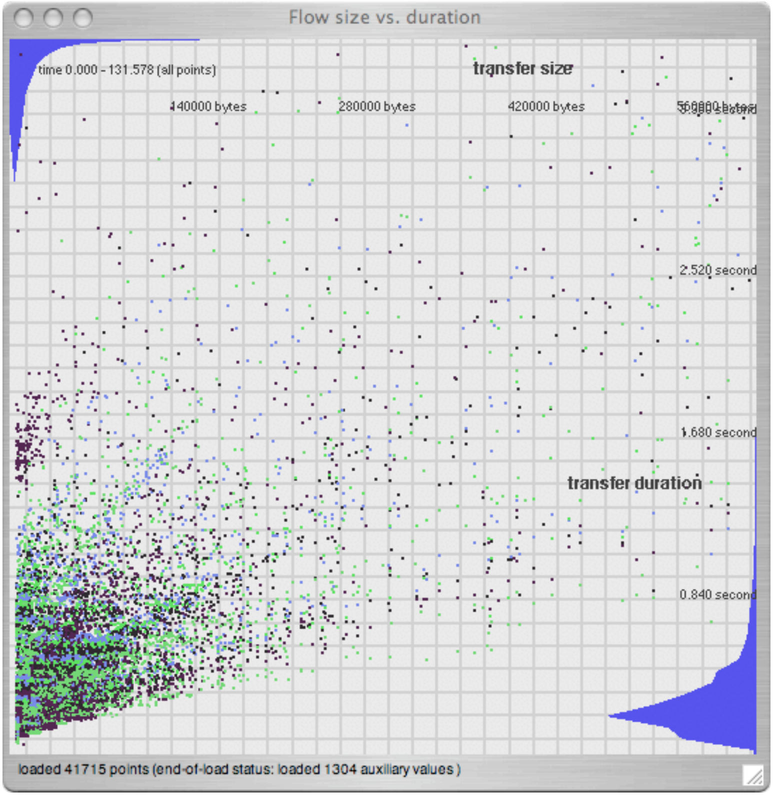


Note that the colors shown are non-standard.




queuing delay
39.726
delay (msec)
time (sec)
0.0
53.4
src/dst list filter
select all
unselect all
change color
value
count
show/hide
Flow size vs. duration
time 0.000 - 131.578 (all points)
transfer size
140000 bytes
280000 bytes
420000 bytes
2.520 second
1.680 second
transfer duration
0.840 second
loaded 41715 points (end-of-load status: loaded 1304 auxiliary values)
SPLAT plot map: Flow size...
SPLAT plot tools: Flow size vs. duration
movie
play/pause
frame window
frame advance
10.000 sec
1.000 sec
frames/sec (0)
10
x grid lines (bytes)
20000
y grid lines (seconds)
0.120
name
shown
active
round-trip time filter
src/dst list filter
queuing delay


src/dst list filter
select all
unselect all
change color
value
count
show/hide
10.52.0.150:10.52.0.130
1654
10.52.0.150:10.52.0.158
1609
10.52.0.150:10.52.0.161
1
10.52.0.150:10.52.0.166
1996
10.52.0.150:10.52.0.194
2183
10.52.0.150:10.52.0.198
1673
10.52.0.162:10.52.0.130
1506
10.52.0.162:10.52.0.158
1330
10.52.0.162:10.52.0.166
1855



In the main scatter plot window, there are certain keys that are recognized for manipulating the plotting space. These keys are treated in a case-insensitive manner.

⟨? h H⟩	Display help text in plotting area
⟨[⟩	Darken background
⟨]⟩	Lighten background
⟨SPC⟩	Play/pause (in movie mode)
⟨.⟩	Step forward one frame in movie mode
⟨,⟩	Step backward one frame in movie mode
⟨g G⟩	Toggle whether grid is shown
⟨/⟩	Toggle whether a diagonal line across plotting space is shown
⟨l L⟩	Toggle display of axis labels
⟨m M⟩	Enter/leave movie mode
⟨o O⟩	Pan to origin
⟨p P⟩	Toggle whether density projections on outer axes are shown

⟨q Q⟩ Constrain display region to square (using smaller of width/height)

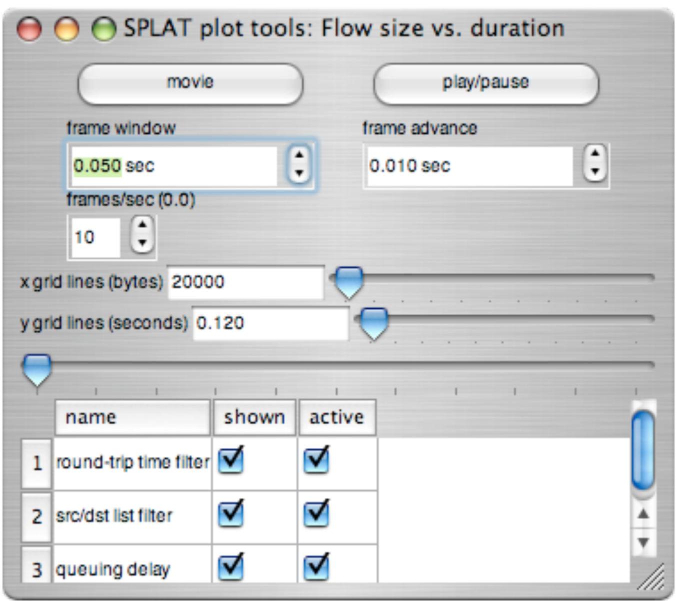
⟨r R⟩ Reset pan/zoom/rotate settings to defaults

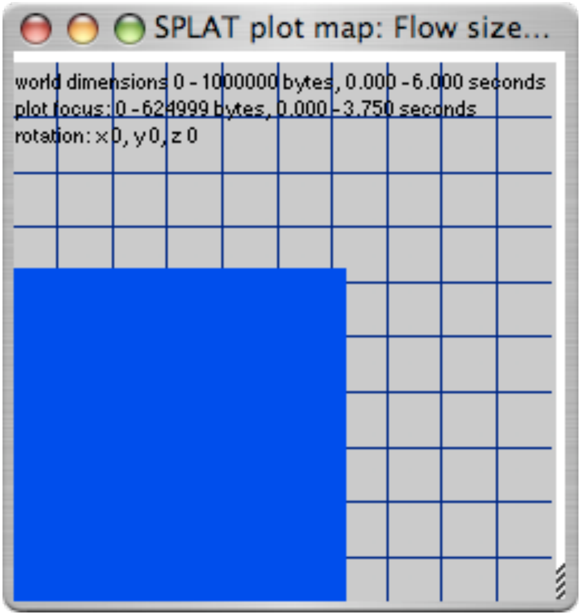
⟨t T⟩ Toggle whether time annotation is shown

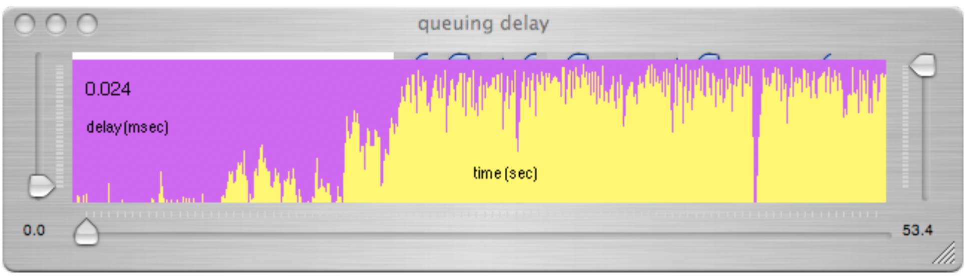
⟨<⟩ Zoom in

⟨>⟩ Zoom out

⟨arrows⟩ Panning




SPLAT plot map: Flow size...
world dimensions 0 - 1000000 bytes, 0.000 - 6.000 seconds
plot focus: 0 - 624999 bytes, 0.000 - 3.750 seconds
rotation: x 0, y 0, z 0


queuing delay
0.024
delay (msec)
time (sec)
0.0
53.4

4 Configuring SPLAT Data Sources

This chapter describes further details of configuring input data for SPLAT. A full reference will be given in an appendix in a future version of this documentation.

4.1 Input Data Format

As noted earlier, SPLAT uses white-space delimited plain text files for input data. Various aspects of the input data stream and plotting enviroment are specified in XML configuration files.

Columns of the input text file are associated with an axis (x, y, or z/time) through tags in the configuration file. There are also specific tags to define the data type for the x and y columns/axes, axis labels, data type precision, and data ranges for x, y, and z/time axes.

Axis data (x and y) can be integer, floating point, or IPV4 address types. Time axis (z) data *must* be floating point data. (Note that integers are automatically cast to float for z/time axis data.) The input data does not have to sorted in time order—it is properly sorted as it is read. Note also that all column number specification is zero-based.

Additional columns may be provided in the input data. These columns may be used by filters, configurable in the XML configuration file (more detail below).

Precision may be specified for any of x, y, or z/time axis data types. For integers, precision is ignored. For floating point values, precision specifies the number of relevant decimal places. For IPV4 addresses, it is the CIDR mask. For example, if our x-axis data is found in column 0 (first column) below and we specify a precision of 16, we effectively have only one unique x-axis value (10.52.0.0). If our z/time axis data is found in column 7 and we specify a precision of 1, there are effectively 3 unique time values (0.0, 0.1, and 0.2).

```
...
10.52.0.190 10.52.0.242 10000 45764 6 7 5747  0.000 0.103591 50
10.52.0.162 10.52.0.250 10000 39019 6 12 13181  0.020 0.109305 35
10.52.0.186 10.52.0.250 10000 39020 6 23 29016  0.068 0.184029 45
10.52.0.162 10.52.0.250 10000 39021 6 26 32403  0.105 0.178167 35
10.52.0.150 10.52.0.242 10000 45765 6 10 10720  0.112 0.077207 25
10.52.0.162 10.52.0.142 10000 50229 6 24 29068  0.131 0.178153 35
10.52.0.158 10.52.0.254 10000 41913 6 25 32351  0.178 0.123730 30
10.52.0.230 10.52.0.250 10000 39022 6 45 62236  0.189 0.424942 70
10.52.0.234 10.52.0.250 10000 39023 6 7 5747  0.210 0.151866 75
10.52.0.162 10.52.0.250 10000 39024 6 64 90924  0.212 0.215752 35
...
```

4.2 Configuration File Format

Splat uses XML configuration files to describe the columnar format in an input file and other key aspects of the input data. Assume that the data example above exists in the (gzip-compressed) file '`flows.txt.gz`'.

```
<splat_data>
    <plot_data filename="flows.txt.gz"
               maxpoints="200000"
               name="Test flow size/dur"
               xcol="6" ycol="8" zcol="7"
               xtype="int"
               xlabel="transfer size" xunits="bytes"
               xrange="0:1000000"
               xprecision="0"
               ytype="float"
               ylabel="transfer duration" yunits="seconds"
               yrange="0:5.0"
               yprecision="3" />

    <filter ftype="list" dtype="int" count="incr"
            col="9" name="round-trip time filter" />

    <filter ftype="distribution" dtype="string" count="6"
            col="0:1" name="src/dst distribution filter" />

    <auxfilter filename="flows_qlen.txt.gz" zlabel="time (sec)"
            wlabel="delay (millisec)" zcol="0" wcol="1" name="queuing de-
lay" />
</splat_data>
```

Each configuration file must have a top-level tag of **splat_data** and exactly one **plot_data** sub-element. The **plot_data** element encapsulates all configuration details pertaining to input file format, data types, etc. There are two other types of elements that may appear at the same level as **plot_data**: **filter** and **auxfilter**. These elements are described below. Note that some elements control similar behavior. When clashing elements are specified behavior is currently unspecified (see the code for what will happen . . .)

4.2.1 plot_data Configuration Element

filename The input file to be read. The input may be gzip-compressed, in which case the final file extension should be '`.gz`'. (required)

name A descriptive name to use for this data set. (optional)

maxpoints The total number of points to load. If this item is omitted, data is loaded until cancelled by the user or until the input file is exhausted. (optional)

begin Specifies the beginning point, in seconds, of the input trace. Rows with time values less than **begin** are ignored. (optional)

end Specifies the end point, in seconds, of the input trace. Rows with time values greater than **end** are ignored. (optional)

[x,y,z]col The zero-based column number in the input data to use for a given axis. (required)

[x,y]type The data type for x and y axes. May be: **int**, **float**, or **ipv4**. (default is **int** if not specified.)

[x,y]label String label for x and y axes. (optional)

[x,y]units String label for x and y units. (optional)

[x,y]precision
Integer-valued precision of x and y data types. For integers, this value is ignored. For floating point types, this is the number of relevant decimal digits. For IPV4 address types, this is a CIDR prefix length. (An alias for IPV4 address types is **mask**.) (optional: defaults to 0 for numeric types and 32 for IPV4 address types)

[x,y,z]range
For x and y axes, defines the "world" extents. No input data is omitted based on these ranges—they are only used to draw the world grid and to provide a coordinate reference. For IPV4 address data types, this range restricts the region one can pan over. For other datatypes, no restriction on panning is imposed. For the z/time axis range, this element is a shortcut for specifying both **begin** and **end** elements. (required for x and y axes, optional for z/time axis)

4.2.2 filter Configuration Element

ftype Specifies the filter type. Must be either **distribution** or **list**. (required)

dtype Specifies the data type of the data specified in **col**. (optional, default to string type)

col Specifies the column number(s) of data to use. These column numbers refer to the input data in the base **plot_data** element. Note that multiple column numbers may be specified, separated by the colon ⟨:⟩ character. (required)

name A descriptive name for the filter. (optional)

count Used primarily for **distribution**-type filters. Specifies a column number containing numeric data that should be counted to create the distribution. A special keyword of **incr** may be used instead of a column number to indicate that the filter should simply count the number of times the data found in **col** was present. (optional)

4.2.3 auxfilter Configuration Element

filename Specifies the file name from which to load time series data. This data may be gzip-compressed, in which case the final file name extension should be '`.gz`'. (required)

name A descriptive name for the time series. (optional)

[w,z]col Zero-based column numbers for x/w and z/time axes. (required)

[w,z]label Descriptive string label for each of x/w and z/time axes. (optional)

4.2.4 Configuration File Example

In the following example, data is loaded from '`spatial.txt.gz`'. The zcolumn is found at column 0 and only data between the 30 second mark and the 630 second mark are loaded. The x and y columns are found in column numbers 1 and 2, respectively, and consist of IPV4 address data. A 16 bit mask is applied to the x and y columns and the full IPV4 space is used as the x-y range. No unit labels are used for the x and y dimensions.

One filter is loaded, using column 4 of the input file '`spatial.txt.gz`'. This column contains integers representing the number of bytes transferred between two IPV4 addresses. A distribution of these values is drawn so that a user can filter the points on the phase plot by transfer amount or a range of transfer amounts.

```
<splat_data>
    <plot_data filename="spatial.txt.gz"
               begin="30" end="630"
               name="Test spatial data"
               xcol="1" ycol="2" zcol="0"
               xmask="16"
               xtype="ipv4addr"
               xlabel="source address" xunits=""
               xrange="0.0.0.0:255.255.255.255"
               ytype="ipv4addr"
               ymask="16"
               ylabel="destination address" yunits=""
               yrange="0.0.0.0:255.255.255.255" />

    <filter ftype="distribution" dtype="int"
            col="4" name="transfer size distribution filter" />
</splat_data>
```

Note that for the above filter, the distribution will consist of single transfer amounts. If one had wished to make a distribution of the *total* amount transferred between a source and destination, one would write something like:

```
   <filter ftype="distribution" dtype="string"
           col="1:2" count="4" name="transfer size distribution filter" />
```

The above filter configuration will essentially use the source and destination addresses (columns 1 and 2) as string hash keys and accumulate the values found in column 4 (the transfer amount). The distribution will be of these total transfer amounts.

5 Using SPLAT: TCP Packet Traffic Example

This chapter will eventually include an example described in our PAM paper along with configuration information and other details.

6 Using SPLAT: Characteristics of Internet Flows

This chapter will eventually include an example described in our PAM paper along with configuration information and other details.

7 Using SPLAT: Spatial Characteristics of Internet Flows

This chapter will eventually include an example described in our PAM paper along with configuration information and other details.

Appendix A Configuration File Reference

This appendix will eventually have full specification on data source configuration files. For now, please refer to the examples in Chapter 4 [Configuring SPLAT Data Sources], page 15.

Appendix B SPLAT Internals: Notes for hacking SPLAT

This chapter will eventually have design notes for SPLAT.

Postscript

The full technical paper describing SPLAT appeared in the Passive and Active Measurement Conference, Adelaide, Australia in March 2006. See http://www.cs.wisc.edu/~jsommers/ for the technical paper, slides, and SPLAT software.

Previous versions of SPLAT have used the (crude) GLUT and the GTK libraries. SPLAT was initially converted to the Qt libraries in 2004 and to Qt version 4 in 2005. Users interested in older versions of the code may contact the author.

Index