

Graphing on the Fly

Generating Real Time Graphs for the Web

Richard Parry, P.E., W9IF

email address:
rparry@qualcomm.com

Home Web Page:
<http://people.qualcomm.com/rparry>

ABSTRACT

This paper describes the steps necessary to create real time GIF formatted graphs for web presentation. The software examples provided are written in perl and make use of the GD and Chart perl modules. Included in the paper are installation instructions, examples showing how to use the modules, and a list of resources for further research. Simple examples are provided as well as a more complex example to show the power and flexibility provided by Chart to create informative and easy to understand graphs. Also included is an explanation which illustrates how a web page can automatically update graphs created on the fly.

March, 1998

INTRODUCTION

If a picture is worth a thousand words, then a graph is worth ten thousand. The ability to represent raw data in a concise and easy to understand manner cannot be overemphasized. Patterns, data subtleties, and other information are often overlooked until they are presented in graphic form. Thanks to computer technology, complex 3D high resolution color graphs provide a wealth of easy to understand information. For real time data collection applications, the data can be made even more useful if the information is also presented in real time (on the fly).

The article, "GIF Images On The Fly", in the November 97 issue of *LJ* described how to create images in real time. The GD perl module described in the article allows the creation of images by specifying graphic primitives such as: lines, polygons, rectangles, arcs, both filled and unfilled. It also supports flood fills, line styling (e.g., dashed lines), horizontal and vertical text rendering; but the package does not directly allow the creation of graphs. Fortunately, several perl modules are available for graph creation. Chart, by David Bonner, is a simple solution for graph development. It is well documented, stable, and easy to use. Since it is a perl module it is particularly easy to implement, modify, and integrate into web applications.

Creating a graph on the fly is only half of the task. The ability of a web browser to request updates from the server automatically is critical for displaying graphs on the fly. After all, the main purpose of making a real time graph is to provide a user with up-to-date information automatically. This means that both the graph and web page need to be continuously updated.

The remainder of this paper explains software installation and provides examples which illustrate how Chart can be used. Lastly, HTML code is included which shows how a web page can be automatically updated to present real time graphs.

INSTALLATION

Chart is a perl module, and like all modules it can be obtained from CPAN (Comprehensive Perl Archive Network). However, since Chart requires GD, also a perl module, let's start with that installation first. Note that perl version 5.003 is required and 5.004 is recommended.

GD can be downloaded from the main CPAN site or any of the numerous mirror sites. The main site is shown in the example. Select the most recent version, which at this writing is GD-1.18.tar.gz.

```
http://www.perl.org/CPAN/modules/by-module/GD
```

After downloading, uncompress and untar GD. If you want to save a few keystrokes, you can do it in one line with: `zcat GD-1.18.tar.gz | tar xvf -`. I like to separate the steps as shown here.

```
gunzip GD-1.18.tar.gz
tar xvf GD-1.18.tar
```

The README file that accompanies the distribution is the ultimate authority for installation instructions. However, with that caveat understood, let's look at the installation process you can expect. For those who have installed other perl modules, the steps should be familiar. The `make test` step is not necessary, but is suggested since it runs regression tests.

```
cd GD-1.18
perl Makefile.PL
make
make test
make install
```

To test the installation, several perl scripts are included in the `demos` directory. Since the output of GD is a GIF formatted file sent to `STDOUT`, it can be displayed using `xview` (`xv` command) as shown here. Note the trailing minus sign, it specifies that the data to be displayed by `xv` is from `STDOUT`.

```
cd demos
perl shapes.pl | xv -
```

With GD installed, we are now ready to install Chart. Since its installation is virtually identical to GD, the steps provided below are without further explanation. Again, the `README` file included with the distribution is the authoritative source for instructions.

```
http://www.perl.org/CPAN/modules/by-module/Chart/chart-0.94.tar.gz
gunzip chart-0.94.tar.gz
tar xvf chart-0.94.tar
cd Chart-0.94
perl Makefile.PL
make
make test
make install
cd samples
xv bars.gif
```

The last step in the above command sequence shows how to use `xv` to display one of the seven graphs generated during installation. The `samples` directory contains GIF examples of all graph types Chart can generate. Both GD and Chart include excellent documentation available with the `perldoc` command.

```
perldoc GD
perldoc Chart
```

CREATING GRAPHS

Chart can create seven types of graphs: points, pie, stacked bars, bars, pareto, lines, and linespoints. It is amazing how little code is necessary to create a useable graph. The two examples shown as Figures 1 and 2 are samples that accompany the software distribution. The source code used to generate the seven example GIF graphs are located in the “`t`” directory. Let’s look at the source code in more detail to gain a better understanding of how to use Chart.

Listing 1 shows the perl script used to create the pareto graph shown in Figure 1. Line 1 causes perl to load the `Chart.pm` module. The following line declares a new object which specifies the type of graph which, in this example, is a pareto graph. Lines 3, 4, and 5 declare the `x`, `y`, and title data respectively. The last line generates the graph as a GIF file with the name `pareto.gif`. Assuming the name of the file is `Listing1.pl`, the graph can be generated on the fly with the following command.

```
perl Listing1.pl
```

Note that by definition, Chart’s API (Application Programming Interface) assumes the first data set is the `X` axis data. All subsequent data sets are `Y` axis values to be graphed. In addition, the API requires that the number of elements in each data set be equal. In the current example, there are six `X` axis points and six `Y` axis values. If additional `Y` axis data sets are to be graphed, those also must have six data points.

Listing 2 shows another simple example; it generates the pie graph shown in Figure 2. There is so little code and it is so similar to the previous example, that I will let the code speak for itself. Now let’s move on to a somewhat more interesting use of Chart.

ADVANCED GRAPHS

Our goal is to create real time graphs, which implies the ability to collect real time data. Examples of real time data are: temperature, wind speed, ocean depth, and stock prices. However, this would be too specific for illustrative purposes. We therefore replace a data acquisition system with a small program that creates random sine and cosine data points. Although this example is contrived, it essentially embodies all the steps necessary for data collection, graph creation, and ultimately making that information available on the web. What makes the example relevant, is that a new graph is generated automatically every 5 minutes. The resulting graph is displayed in Figure 3. In the following section, we will discuss the design of the web page that is automatically refreshed to insure the graph shown is always the most recent version.

Listing 3 shows the perl code used to generate the graph on the fly. To understand the more advanced features of Chart, let's examine the source code in detail. Like the previous examples, the lines in the listing are numbered for illustrative purposes only.

The first line of significance in the listing is line 10, where the Chart perl module is declared. To insure that a new graph is generated each time the program is executed, random variables in lines 17 to 20 specify the amplitude and frequency of the sine and cosine functions. Line 16 is relevant since it specifies a positive offset to insure that negative values are not generated since the present version of Chart does not support negative values. The "TO DO" list shows that future versions of Chart should support this feature along with 3D graphs.

The `for` loop starting on line 23 creates the data arrays which hold the raw data to be graphed. On line 29, the Chart object type and size are declared using the `new` method. For the present we are interested in generating a simple graph of type "Lines" with width and height of 600 by 250 pixels respectively.

Lines 32 through 34 call the `add_dataset` method and pass the calculated values previously stored in the arrays. The first array, `radarray`, provides the X axis data (angle in radians). The remaining arrays, `sinarray` and `cosarray`, represent the data to be plotted for each of the two transcendental functions.

Chart now has all the information necessary to plot the data. However, let's add some amenities for clarity and to make the graph more readable. Since we are generating a real time graph, we need to provide a way to specify the time when the graph was generated. The graph is time and date stamped on line 37 and later passed to Chart for graphing on line 41 using the `set` method and hash key `sub_title`.

For most graphs, the X axis contains a large number of data points. For example, our graph contains 100 points. It would not be possible to display all X axis values. Therefore, Chart uses the `set` method and `custom_x_ticks` hash key to specify exactly which X axis labels are to be displayed. This is done on line 38, where we declare the `@xticks` array and initialize the 10 values we are interested in plotting along the X axis. This information is passed to Chart on line 46.

The legend labels are declared in an array on line 39 and passed to Chart on line 44 using the `set` method and hash key `legend_labels`. The title and axis labels are declared on line 40, 42, and 43, again using the `set` method.

The border that surrounds the graph can be modified from its default value of 10 pixels. In the example, the border is set to 1 using the `set` method and `graph_border` key as shown on line 47. The horizontal and vertical grid lines are drawn by setting the key `grid_lines` to true as shown on line 48. Initialization is now complete and the graph is generated on line 50. Note that the full directory path for the GIF file is declared. As we shall see in the next section, this perl script will be called as a cron job, and as such, the graph will be created in the default directory. In most cases this is not where we want it and therefore we must provide the full path.

Chart supports additional features such as the ability to specify a transparent background which is useful in web applications. One of the more interesting features is the ability to let Chart sort data in ascending or descending order before plotting. Specifying the colors for each of the datasets and other amenities are also provided and we leave those features for investigation by those having additional requirements.

UPDATING THE GRAPH

Each time the perl program in Listing 3 is executed, a new graph is generated. We therefore need a mechanism to automatically execute the program. Normally the updating process is application specific. For our current example, executing the program as a cron job is a simple solution. Using the `crontab -e` command, and adding the line shown below to the crontab file will cause the program to be executed every five minutes. Recall that on line 50 of the source listing the complete file path for the GIF file was declared. For similar reasons the full path for the perl script must be declared as shown below. Therefore you will have to alter the file path from that shown to match the location of the script on your system. Note that the cron job is a single line and is shown here as multiple lines using the backslash character.

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * \  
/home/httpd/html/iweb/graphonfly/Listing3.pl \  
>/dev/null 2>&1
```

STDOUT (standard out) and STDERR (standard error) output are redirected so that the process can be executed as a background task. In other words, we want our job to run in the background and don't want output to be sent to a window. Using the command line options shown, output will be sent to `/dev/null` which acts as a "black hole" for data.

UPDATING THE WEB PAGE

We now have a new graph generated every five minutes and need to develop a web page to display it. Writing a standard web page without any consideration to the unique real time requirements of our application would require the user to manually "reload" the page every five minutes, clearly not a realistic solution. In addition, as we shall soon see, a simple reload does not guarantee a refreshed web page.

Just as we developed a means to update the graph automatically by making a cron job, we need a method to force the web browser to reload the page automatically. This is referred to as a "client pull" operation, since we are asking the client (web browser) to pull new data from the server. To do this, we need to overcome the web browser's, and in some cases, a proxy server's cache which keep copies of the page. Since a browser keeps a cache of recently visited pages locally, after five minutes the graph displayed will be out-of-date. Fortunately, HTML provides a META tag for this purpose. META tags are unique in two respects. First, the META tag is not used in pairs (there is no closing tag such as `</META>`). Second, the tag must be placed within the HEAD portion of the HTML code. An examination of Listing 4 shows the "Refresh" tag that causes the page to automatically reload (pull) every 300 seconds (5 minutes).

```
<META HTTP-EQUIV="Refresh" CONTENT="300">
```

It would appear we have a solution for obtaining a fresh copy of the graph every 5 minutes. Unfortunately, life is not so simple. The refresh tag does not guarantee that the browser will not use its cache. In addition, if a proxy server is used, the above solution is also unlikely to be sufficient. Since the proxy server has a copy of the page, it may resend the same page to the browser rather than getting a new copy from the originating web server. The result is that we do not get a new copy of the graph with a simple web browser reload.

For those not familiar with proxy servers, a slight digression is warranted. Assume you are at a large university with tens of thousands of students all accessing the same page. To decrease Internet traffic, a proxy server is often installed between the university LAN and the Internet which keeps a local copy of recently accessed pages. In this way, when a student needs access to a commonly used page, it is immediately available locally from the proxy server. The bad news is that a simple reload by the local browser may not provide a new copy to the user from the source web server. Fortunately, HTML provides another META tag. An examination of the listing shows syntax for the "Expires" tag.

```
<META HTTP-EQUIV="Expires" CONTENT="Wed, 11 Feb 1998 04:35:12 GMT">
```

This line added to the HTML code informs the browser that the information is stale after the date specified. As long as the tag specifies a time in the past, the web browser interprets the page as being expired and requests a new copy during the next reload request. Although we are not interested in a specific time for this example, for the sake of completeness, note that the time must be in GMT format. The UNIX `date -u` command can be used to provide that information.

The two META tags discussed so far should suffice to assure us of an up-to-date web page. Unfortunately some browsers ignore the "Expires" tag. For those cases, another META tag can be used to prevent caching. The "pragma" tag below asks the web server to send a request to the browser to not cache the page. In this case we are relying on the server, rather than the browser, to provide the functionality we desire. Again, not all web servers properly support this tag.

```
<META HTTP-EQUIV="pragma" CONTENT="no-cache">
```

Despite the use of these three tags to control caching, updating of the web page is not guaranteed. In practice, different browsers and servers support these tags better than others. In some cases I have found that a simple reload may only reload the web page text and not the GIF image, or in other cases it may not update the page at all. In fact, different platform versions of the same browser may also act differently. An exhaustive comparison is not possible, but Microsoft's Internet Explorer 4.0 for NT is one browser that appears to function properly in real time web updating applications (your mileage may vary).

A REAL, REAL TIME GRAPH

The three previous examples were intended to show how to generate graphs on the fly by using examples that can be easily understood and reproduced without additional hardware or sophisticated software. However, to help give the reader an appreciation of how Chart can be used in real world applications, Figure 4 is provided. The graph shows network traffic from a wireless LAN network. This is a more advanced project since it requires the use of logs to store data. It also requires a more advanced mechanism for keeping track of time. Nonetheless, despite its increased complexity, if the reader has a firm understanding of how the previous simple examples work, developing a real world, real time application is not much more difficult. The *Resources* section included in this paper shows other real world uses for graphing on the fly, although not all use the Chart perl module.

CONCLUSION

There is little doubt that a graph is the best method for presenting large amounts of data in a form that can be easily understood. Many graphs are static, they are the result of a single data collection activity. However, when graphs are dynamic and the data collection process is continuous, then displaying the data must also be continuous. Chart provides an easy to implement solution for creating graphs on the fly for those applications. Integrating the dynamic graph into a web page for the world to see provides us with a vehicle for the dissemination of vast amounts of data in a timely and informative manner.

RESOURCES

1. CPAN Archives: <http://www.perl.com/CPAN>.
2. Ball, Jimmy, "GIF Images on the Fly," *Linux Journal*, November 1997, pp. 48-50.
3. Lachiniet, Mark, "Linux at Holt Public Schools," *Linux Journal*, September 1997, pp. 34-36.
4. Elton, Peter, "Linux as a Proxy Server," *Linux Journal*, December 1997, pp. 14-20.
5. The GD perl module used in this paper is located at:
<ftp://www.perl.org/pub/CPAN/modules/by-module/GD/GD-1.18.tar.gz>
6. The Chart perl module used in this paper is located at:
<ftp://www.perl.org/pub/CPAN/modules/by-module/Chart/chart-0.94.tar.gz>
7. HTML Meta tags are discussed in RFC 1886, Hypertext Markup Language - 2.0, November 1995, available at:
<http://hegel.ittc.ukans.edu/topics/internet/rfc/rfc18xx/rfc1866.txt>
8. An interesting real time web page that shows Pacific Ocean tides at the Scripps Institution of Oceanography, San Diego, California is located at:
<http://www.willy.com/Scripps>
9. Columbia University in New York City, provides guidelines to its students for using its secure web server. These guidelines provide information about the web browser cache which is a crucial part of this paper. The URL is:
<http://www.columbia.edu/acis/rad/secure-server/publishing.html>

Listing 1 Perl code used to generate the Perato Graph in Figure 1

```
1 use Chart::Pareto;
2 $g = Chart::Pareto->new;
3 $g->add_dataset ('bifur', 'bofur', 'bombur', 'fili', 'kili', 'nili');
4 $g->add_dataset (6, 4, 9, 2, 7, 13);
5 $g->set ('title' => 'Pareto Chart');
6 $g->gif ("pareto.gif");
```

Listing 2 Perl code used to generate the Pie Graph in Figure 2

```
1 use Chart::Pie;
2 $g = Chart::Pie->new;
3 $g->add_dataset ('foo', 'bar', 'junk');
4 $g->add_dataset (6, 4, 9);
5 $g->set ('title' => 'Pie Chart');
6 $g->gif ("pie.gif");
```

Listing 3 Perl code to generate the graph in Figure 3

```

1  #!/usr/bin/perl -w
2
3  # Description:  A perl script to generate a GIF graph at 5 min.
4  # intervals.  The graphs are sine and cosine functions with random
5  # amplitude and frequency parameters. The values are placed in an
6  # array and the graph generated.  The graphs may be seen using xv
7  # or the html code provided in Listing 4.
8
9  use strict;
10 use Chart::Lines;
11
12 my($i, $graph, @sinarray, @cosarray, @radarray,
13     $offset, $sinamp, $cosamp, $sinfreq, $cosfreq);
14
15 # Compute random variables for this run
16 $offset = 5;
17 $sinamp = rand(5);
18 $cosamp = rand(5);
19 $sinfreq = rand(5);
20 $cosfreq = rand(5);
21
22 # Create arrays to be used as data for graph
23 for ($i = 0; $i < 10; $i += 0.1) {
24     push @sinarray, $offset + $sinamp * sin($sinfreq * $i);
25     push @cosarray, $offset + $cosamp * cos($cosfreq * $i);
26     push @radarray, int($i);
27 }
28
29 $graph = Chart::Lines->new(600,250);# declare object
30
31 # Pass data for graphing
32 $graph->add_dataset (@radarray);
33 $graph->add_dataset (@sinarray);
34 $graph->add_dataset (@cosarray);
35
36 # Add the amenities
37 my($LastUpdate) = scalar localtime;
38 my(@XTicks) = qw(1 11 21 31 41 51 61 71 81 91);
39 my(@GraphLegends) = qw(Sine Cosine);
40 $graph->set ('title' => 'Graphing on the Fly (5 min.)');
41 $graph->set ('sub_title' => "Last Update $LastUpdate PST");
42 $graph->set ('x_label'=> 'Angle in Radians');
43 $graph->set ('y_label'=> 'Amplitude');
44 $graph->set ('legend_labels' => \@GraphLegends);
45 $graph->set ('stagger_x_labels' => 'false');
46 $graph->set ('custom_x_ticks' => \@XTicks);
47 $graph->set ('graph_border' => 1);
48 $graph->set ('grid_lines' => 'true');
49
50 $graph->gif ("/home/httpd/html/iweb/graphonfly/Figure3.gif"); #Make graph

```

Listing 4 HTML code for the Web page shown in Figure 3

```
<HTML>
<HEAD>
<TITLE>Graphing on the Fly</TITLE>
<META HTTP-EQUIV="Refresh" CONTENT="300" >
<META HTTP-EQUIV="Expires" CONTENT="Wed, 11 Feb 1998 04:35:12 GMT">
<META HTTP-EQUIV="pragma" CONTENT="no-cache">
</HEAD>

<BODY BGCOLOR=#FFFFFF TEXT=#000000>
<CENTER><H2>Graphing on the Fly</H2></CENTER>
This page is automatically updated at 5 minute intervals. The graphs
are generated using perl modules Chart.pm and GD.pm. Chart can
easily generate seven types of graphs and provides additional
functionality to insure that raw data is presented in an easy to unde
rstand manner.
</a>
</BODY>
</HTML>
```

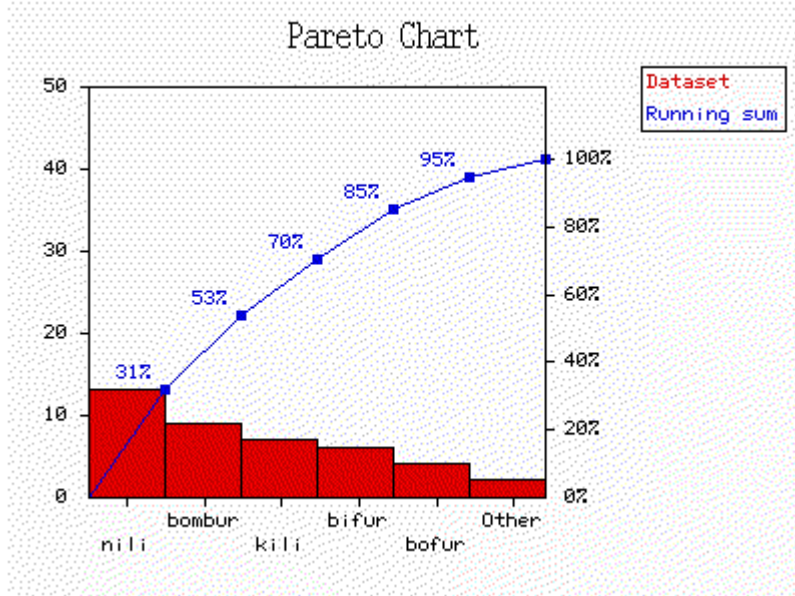


FIGURE 1

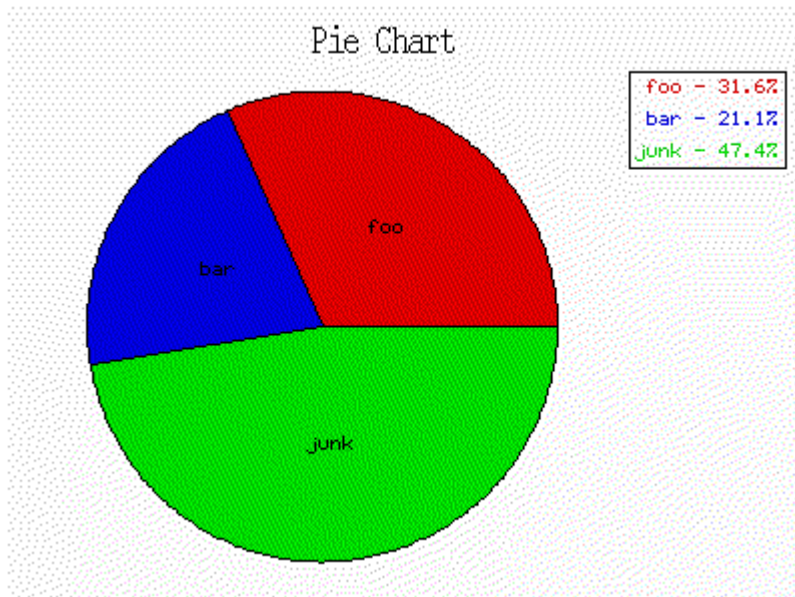


FIGURE 2

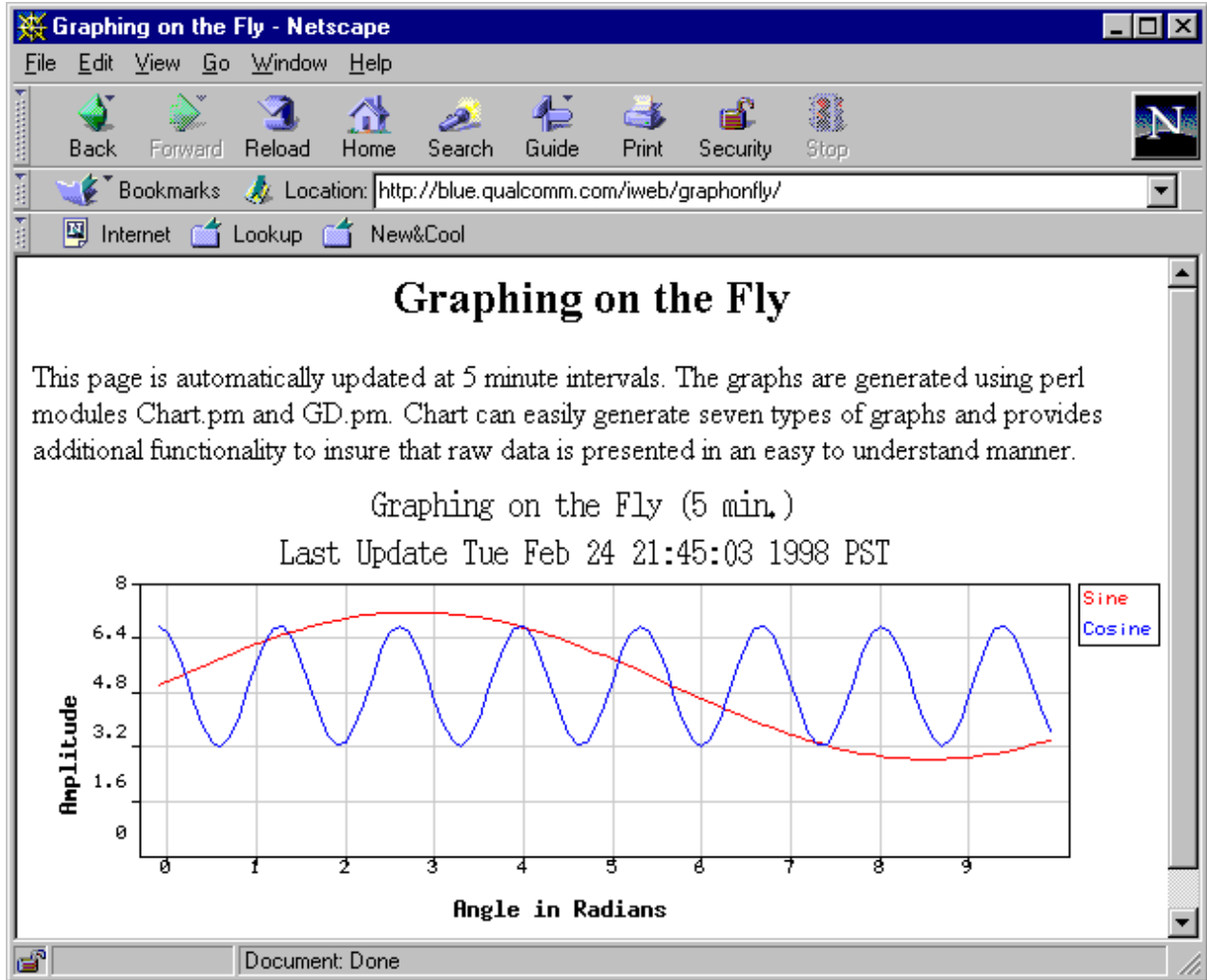


FIGURE 3

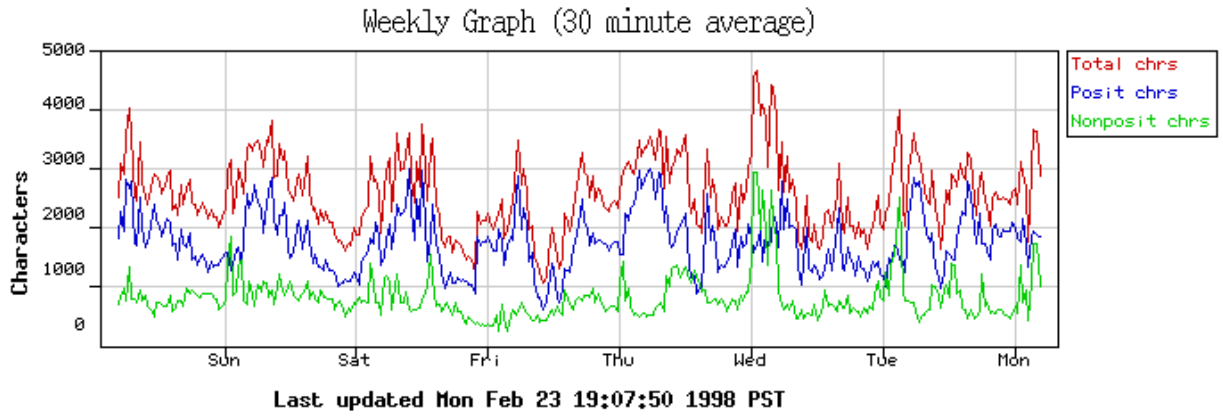


FIGURE 4



Richard Parry is a software engineer at Qualcomm, Inc., known by most as the home of the email program Eudora. When not sitting in front of a monitor, he plays racquetball, but does entirely too much of the former and not enough of the latter. He is presently working on getting a life. He can be reached at rparry@qualcomm.com or you may visit his home page at <http://people.qualcomm.com/rparry>.