Spectral Analysis of Network Traffic

Georgi T. Todorov

September 12, 2007

Introduction

Computer networks are part of every company and business. As more and more important data is moved around these networks, their security is becoming even more important. There are various tools that analyze network traffic and look for different patterns and "fingerprints" to identify malicious traffic among other things. This study uses a technique that is popular in the area of sound waves and engineering. That technique uses Fourier Transform - the spectral function [Tolstov, 1976] - to generate the periodicity or frequency of given set of data.

In mathematics, the Fourier transform is a certain linear operator that maps functions to other functions. Loosely speaking, the Fourier transform decomposes a function into a continuous spectrum of its frequency components, and the inverse transform synthesizes a function from its spectrum of frequency components. A useful analogy is the relationship between a series of pure notes (the frequency components) and a musical chord (the function itself). In mathematical physics, the Fourier transform of a signal x(t) can be thought of as that signal in the "frequency domain." This is similar to the basic idea of the various other Fourier transforms including the Fourier series of a periodic function.[Wikipedia, 2007b]

Suppose x is a number—complex-valued Lebesgue integrable function. The Fourier transform to the frequency domain ω is given by the function:

$$X(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t) e^{-i\omega t}\, dt, \ \textit{for every real number } \omega.$$

When the independent variable t represents time (with SI unit of seconds), the transform variable ω; represents angular frequency (in radians per second). If $X(\omega)$ is defined as above, and $x(t)$ is sufficiently smooth, then it can be reconstructed by the inverse transform:

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} X(\omega) e^{i\omega t}\, d\omega,\ for\ every\ real\ number\ t.$$

The interpretation of $X(\omega)$ is aided by expressing it in polar coordinate form, $X(\omega) = A(\omega)e^{i\phi(\omega)},\ where:$

$$A(\omega) = |X(\omega)|\ the\ amplitude \phi(\omega) = \angle X(\omega)\ the\ phase$$

Then the inverse transform can be written:

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} A(\omega) e^{i(\omega t + \phi(\omega))}\, d\omega$$

which is a recombination of all the frequency components of $x(t)$. Each component is a complex sinusoid of the form $e^{i\omega t}$ whose amplitude is proportional to $A(\omega)$ and whose initial phase angle (at t = 0) is $\phi(\omega)$.

The implementation of the Fast Fourier Transform that is used is called "Fastest Fourier Transform in the West" (FFTW). It is known as the fastest free software implementation of the Fast Fourier transform (FFT) algorithm. FFTW can compute transforms of arbitrarily sized arrays (including prime lengths). It works best on arrays which have sizes which have multiple factors, and uses a variety of algorithms to compute the transform. A particularly strong feature is the ability to run tests on a particular sized array on a particular processor. These tests determine the fastest method to compute the transform, and this knowledge, which the authors call "wisdom" can be stored in a file or string for later use. Given a large number of transforms to be computed, all of the same size, this method can yield a worthwhile gain in speed compared to the default case when FFTW uses a heuristically chosen algorithm.[Wikipedia, 2007a]

Fourier Transform and Network Traffic

Given the nature of network traffic data, Fourier Transforms can be used to identify periodicity over the network. Periodic traffic, in general, is special traffic over the network and should be of interest. When optimizing a network, it is interesting to know how often backups are being performed and does this traffic overlap with other periodic traffic (server synchronization for example). If so, backups can be shifted to different time to distribute the load over time instead of creating peak

and low times. Also, a periodic traffic could mean a probing script, or a hacker that comes back around the same time every day.

For this study a real data sample was used. The data consist of one day of incoming traffic with 1 second samples (86400 samples total). First the data was gathered using libstatgrab[i scream, 2006] and processed to an rrd database using rrdtool:

```
rrdtool create traffic.rrd -s 1 -b 1169916192
DS:in:DERIVE:1000:0:12500000
DS:out:DERIVE:1000:0:12500000
RRA:AVERAGE:0.5:1:86400
RRA:AVERAGE:0.5:7:86400
```

The gathered data was plotted like this:

```
rrdtool graph data3.1.png -s 1170543274 -e 1170629672 -v "Traffic"
–lazy -h 80 -w 6000 -l 0 -a PNG
DEF:inboundorig= traffic.rrd:in:AVERAGE
LINE1:inboundorig#FF00FF:"Incoming traffic"
```

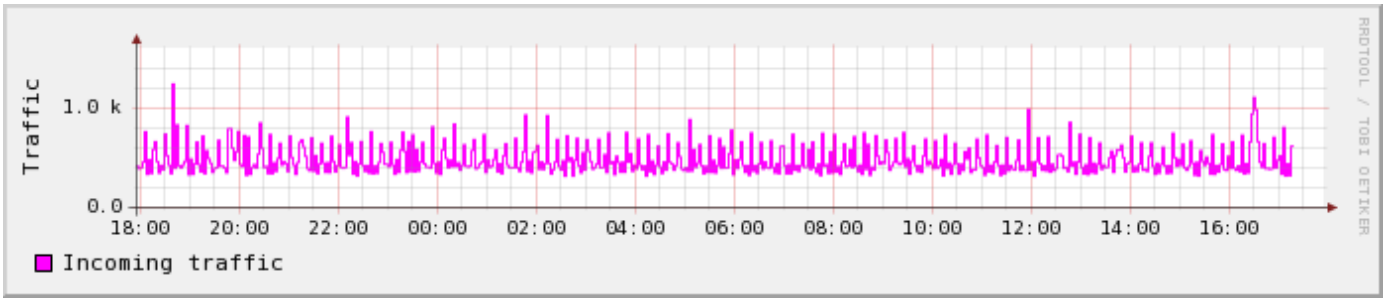

Figure 1: Original Traffic Flow

The actual implementation of the Fourier Transforms is the FFTW3 library[Frigo and Joh which provides forward and inverse transform options. For our ease a real to real transform was performed. The input (list of doubles) was extracted from the rrd database using the following set of commands:

```
rrdtool fetch traffic.rrd AVERAGE | perl stdin.pl | cut -d " " -f 1,2 > data.in
```

Where stdin.pl is a simple perl script that converts the numbers from their scientific annotation to normal annotation that will be processed by the main program. Also in the above example, the original timestamp is also recorded and processed even though it is never used later.

The next tool used is a simple C++ program that uses the FFTW3 library to compute the transform of the traffic. The output of the program is in data format that can be used by gnuplot[Thomas Williams, 2004].

```
#include <fftw3.h>
#include <iostream>
#include <vector>
#include <math.h>

using namespace std;
int main ()
{
  int size=0;//number of samples
  double cor = 0.00001157; // equals to 1/86400 of the second
  long temp2; // timestamp
  double temp,; // data
  vector<double> data; // data vector
  vector<long> times; // timestamp vector
        double absval; // compute absolute value for better plotting
        int m=1;//a counter
        double cc; // frequency count

  while (cin >> temp2){
    times.push_back(temp2);
    cin >> temp;
    data.push_back(temp);
  }

  size = data.size();
  double original[size]; // input array
  double transformed [size]; //output array
  //create an array with the data for fftw_plan_r2r_1d

  for ( int i = 0; i< size; i++ ){
    original[i] = data[i];
  }

  fftw_plan p;
  p = fftw_plan_r2r_1d( size ,  original , transformed, FFTW_R2HC , FFTW_ESTIMATE );
  fftw_execute(p);
  // output data for gnuplot
  for (int i = 0;i < size-2;i++){
    absval = sqrt(transformed[i]*transformed[i]);
```

```
    cc = (double)m*cor;
          cout << cc<<" "<<absval<< endl;
    m++;
  }
  return 0;
}
```

After the data was processed, using the following lines, a readable spectrum images was generated under OS X using AquaTerm[Persson, 2004] with clear indications of periodicity in the sample data.

```
gnuplot> set term aqua
gnuplot> set size 1,1
gnuplot> set style data line
gnuplot> set xlabel "Freqency" font "Curier,14"
gnuplot> set title "Fast Fourier Transform" font "Curier,14"
gnuplot> set grid xtics ytics
gnuplot> set yrange [0:30000000]
gnuplot> set xrange [0:0.3]
gnuplot> set xtics 0.025
gnuplot> plot "data.dat"
```

Analyzing

Once the data is being transformed and plotted it is time to read the image. The usual way of plotting the transform is with the lower frequency to the left (with an absolute minimum at the origin) and grows to the right. In our case we have one day of data which is 86400 seconds, we also have 86400 entries, so our limits are 1/86400 to the left - meaning that an even can repeat at the rate of once every second - and 1 to the right - which means that the event occurs once in the data (or once a day). In our example only a third of the actual spectrum is displayed, since the most interesting part to look at is the closer to the left, where events are more frequent. One hertz in our case means "once every day", but since we do not have data for more than one day, the result is not accurate. 0.3 of a hertz is "1/3 of a day" in our case which is "once every 8 hours". So in our one day data we will look at events that repeat at least once every 8 hours and will emphasize on the left part of the plot (every few minutes). Looking at Figure 2 it is clear that there are a lot of repeating events. We can say several facts about this traffic. Starting at about 0.016 (23min) and going to the right of the image we can identify several different events with frequency change of about 0.024(36mins) apart. That is quite interesting. The first event is at frequency 0.016Hz and it means that there is an event throughout the day that generates the same amount of traffic every 23 minutes. Since the data was harvested on a web server, it could be a spider that is refreshing his index. We cannot say what caused the traffic, but we can say that something is generating traffic every 23 minutes.

One factor to consider here is that the analyzed data is too little for the transform to represent accurate frequent events in the traffic, especially when they get more to the right of the graph.

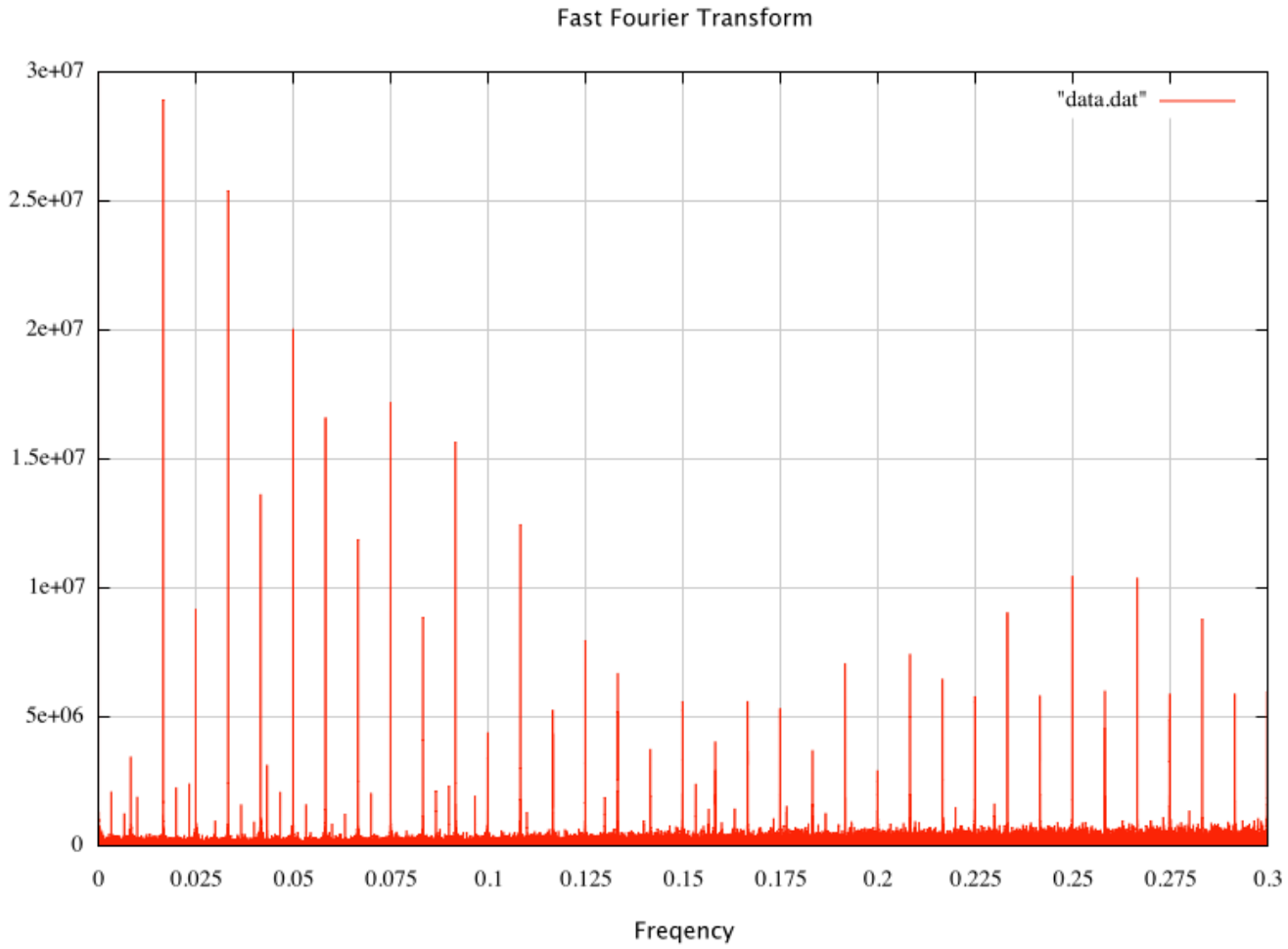

Figure 2: 1/3 of the spectrum

In order for our analysis to be accurate we must look at the leftmost part of the spectrum, where events are sampled many times in one day and are detected correctly. Most events that are more than couple of tens of minutes could be just regular traffic generated on the network. A higher resolution image of the part from the origin to just before the first big spike could be the best place to look at. That is between 0 and 0.0165. Using the following lines we plot the data and we can see on Figure 3 that there is a pretty nice peak at around 0.0083. Looking at the that spike in the graph, we can certainly say that there is a good chance that an event is generating a good amount of traffic every about 12 minutes. Looking at the actual traffic graph, however, it is very hard to tell weather this is a separate event or a part of an event that repeats alot faster. we can see that there are maybe 3 actual events in total that can be stated as repeating throughout the day and the rest is just noise. So here something strange appears. There are many spikes in the plot that are not really in the original data.

```
gnuplot> set term aqua
gnuplot> set size 1,1
gnuplot> set style data line
gnuplot> set xlabel "Freqency" font "Curier,14"
gnuplot> set title "Fast Fourier Transform" font "Curier,14"
gnuplot> set grid xtics ytics
gnuplot> set yrange [0:4000000]
gnuplot> set xrange [0:0.016]
gnuplot> set xtics 0.0013
gnuplot> plot "data.dat"
```

When the issue was discussed with my advisor, he suggested to smoothen the data and to plot it again, hoping for cleanup of most spikes, but the result was not much better. The plot still looked like the one on Figure 2. The smoothing of the data was done using the formula:
Given original values t1, t2, t3, t4 ... compute the new corresponding smooth values s1, s2, s3, s4 as follows:

$$s1 = t1$$
$$s2 = t2 + s1/2$$
$$s3 = t3 + s2\ /2$$
$$s4 = t4 + s3\ /2$$

The next step in trying to identify events was to introduce an artificial traffic event that generates huge traffic every 20 minutes. In theory this event should produce a clear spike in the 20 minute zone on the graph ($1/(20*60) = 0.00083333$Hz). And indeed, after plotting the modified data, the spike did appear on the 0.0008xxxx spot. Unfortunately it appeared on any frequency multiple of 0.00083xxx as shown on Figure 4.
Here is the code to modify the data:

```
my $temp=0;
while (<STDIN>){
        chomp $_;
```

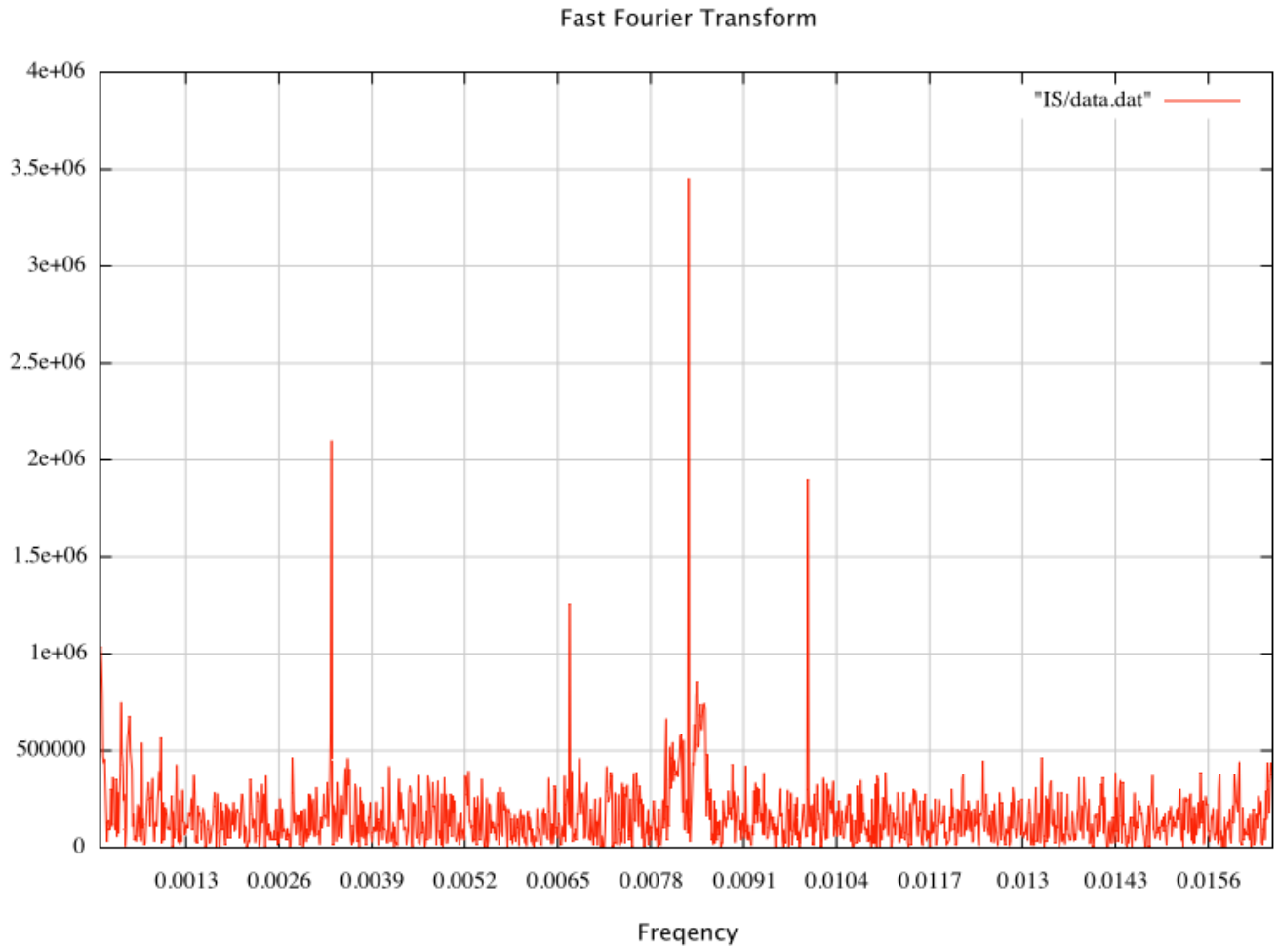

Figure 3: A small part of the most frequent events

```
        @a = split/\s+/, $_;
        $asd=$a[0];
        printf "%.2f", $asd;
        print "\n";
        if ( $temp%1200 eq 0 ){
                print "100000000\n";
        }
        $temp++;
}
```

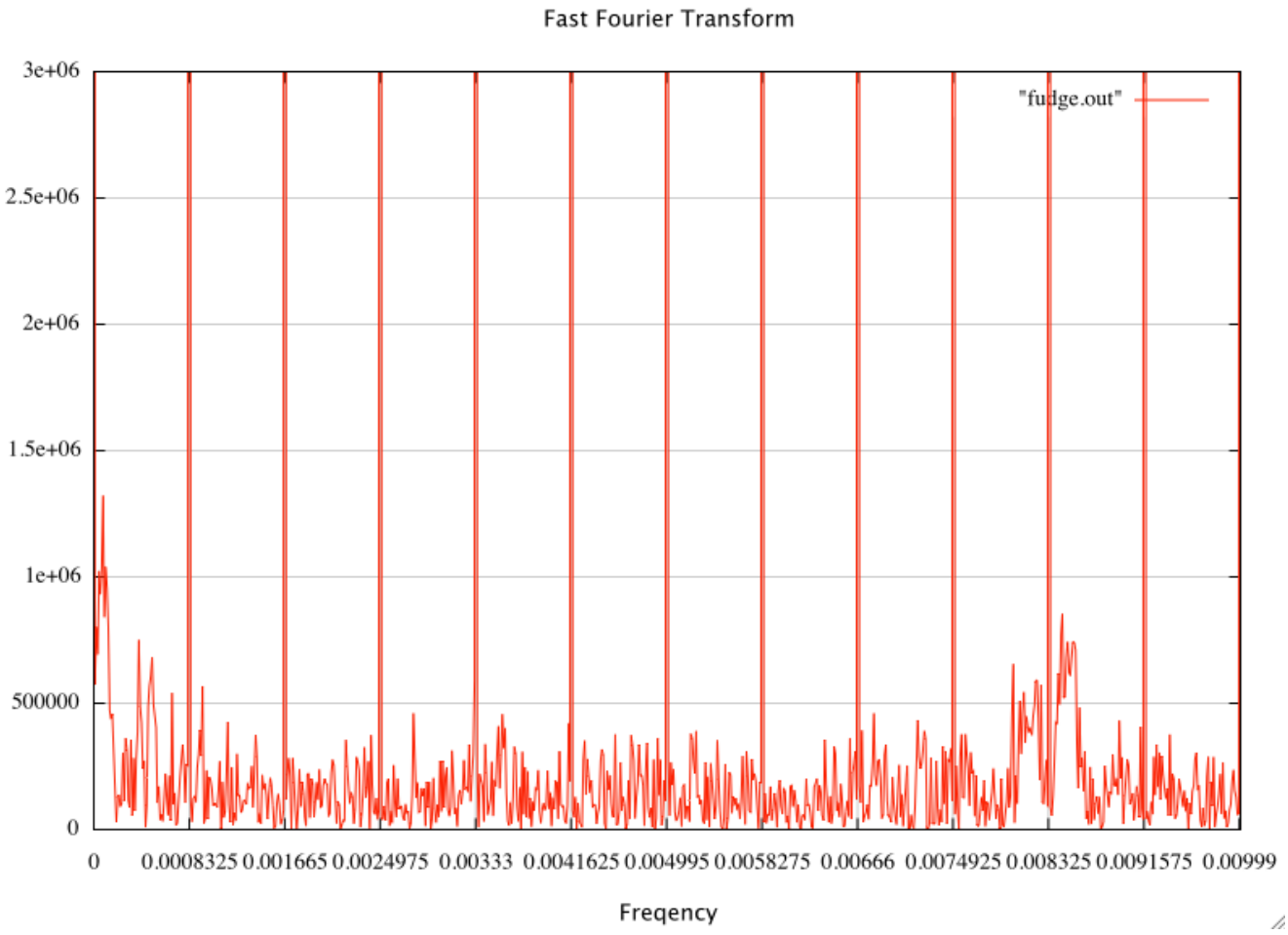

Figure 4: 0Hz to 0.1Hz artificial 20 minute event

The result in the end is a little disappointing. It does not produce the expected output, and just looking at the pure output it is very hard to identify clean events from the output. The nature of network traffic, even appropriate for fourier transforms is too random and with lots of noise that interferes with the traffic generated from regular basis events. The expected result was that a clean cut spikes will be produced like when producing a fourier transform of the function $sin(x)$. Instead the randomness of the network traffic as a function produces too much noise in the output and makes the technique not very practical. Other techniques such as wavelets have been much

more successful in analyzing the flow of network data. It is clear, however, that applying Fourier Transforms to network traffic does not produce spectrum with clean information on it.

Bibliography

[Frigo and Johnson, 2006] Frigo, M. and Johnson, S. G. (2006). Fastest Fourier Transform in the West. `http://www.fftw.org`.

[i scream, 2006] i scream (2006). libstatgrab. `http://www.i-scream.org/libstatgrab/`.

[Kaplan, 2002] Kaplan, I. (2001,2002). A Notebook Compiled While Reading Understanding Digital Signal Processing by Lyons. `http://www.bearcave.com/misl/misl_tech/signal/index.html`.

[Persson, 2004] Persson, P. (2001-2004). AquaTerm.app, your friendly plotting front-end. `http://aquaterm.sourceforge.net`.

[Thomas Williams, 2004] Thomas Williams, C. K. (1986 - 1993, 1998, 2004). GNUPLOT: An Interactive Plotting Program. `www.gnuplot.info`.

[Tolstov, 1976] Tolstov, G. (1976). *Fourier Series*. Courier Dover Publications.

[Wikipedia, 2007a] Wikipedia, t. f. e. (2007a). FFTW. `http://en.wikipedia.org/wiki/FFTW`.

[Wikipedia, 2007b] Wikipedia, t. f. e. (2007b). Fourier transform. `http://en.wikipedia.org/wiki/Fourier_Transform`.

[www.captain.at, 2005] www.captain.at (2005). Spectrograms with the FFTW FFT-Spectrograph library and GNUplot on Linux. `www.captain.at`.