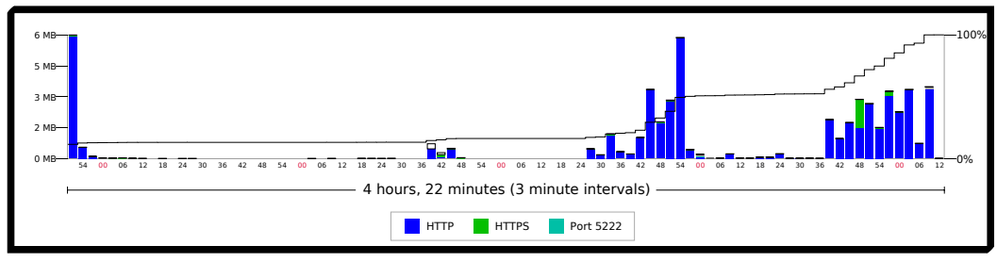


Forensics Visualizations with Open Source Tools

Simson L. Garfinkel, Ph.D

Associate Professor, Naval Postgraduate School

http://www.simson.net/

Tuesday, November 5th, 2013, 8:40am - 9:15am

"The views expressed in this presentation do not necessarily reflect those of the Department of Defense or the US Government."

Forensic visualizations serve two purposes: Presentation & Discoverery

Presentation — visualizations can explain data

- Report or Courtroom
- Summarize data
- Present time series information — illustrate a sequence of events
- Provide iconic representation of an idea

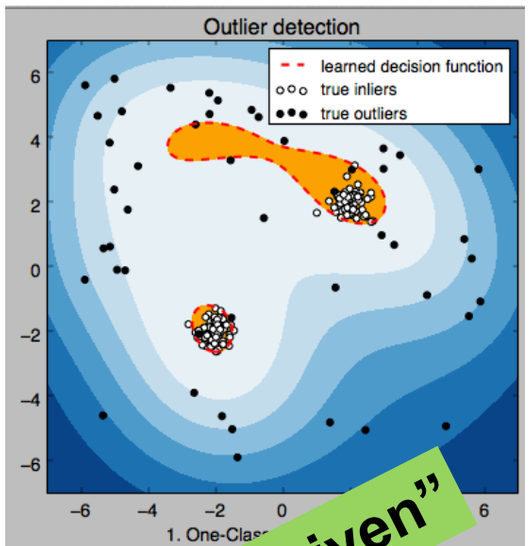
  —*This seems to be what most forensic visualizations are used for*

"Hand-drafted"

Discovery — visualizations that help us learn something

- "Situational awareness"
- Network connections
- Summarization
- Correlation

  —*This is what we would like to use forensic visualizations for.*

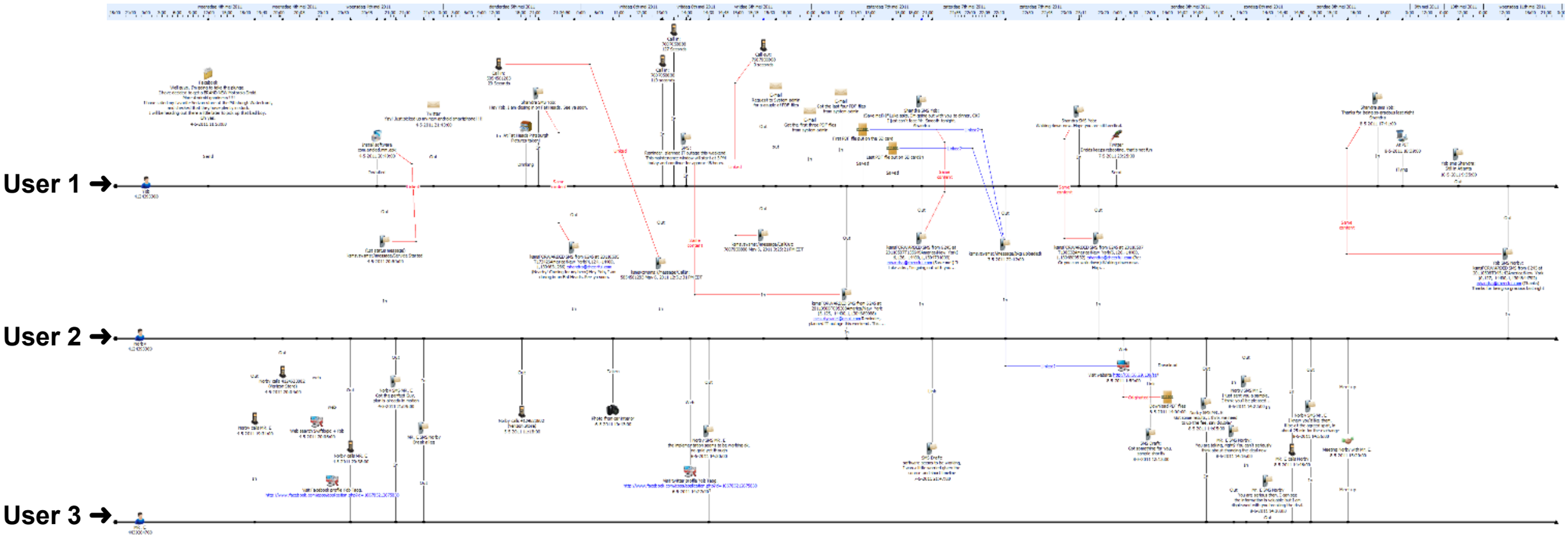


"Data-driven"

Presentation visualizations:
Illustrate complex concepts or sequences of events

DFRWS 2011 Solution, FoxIT

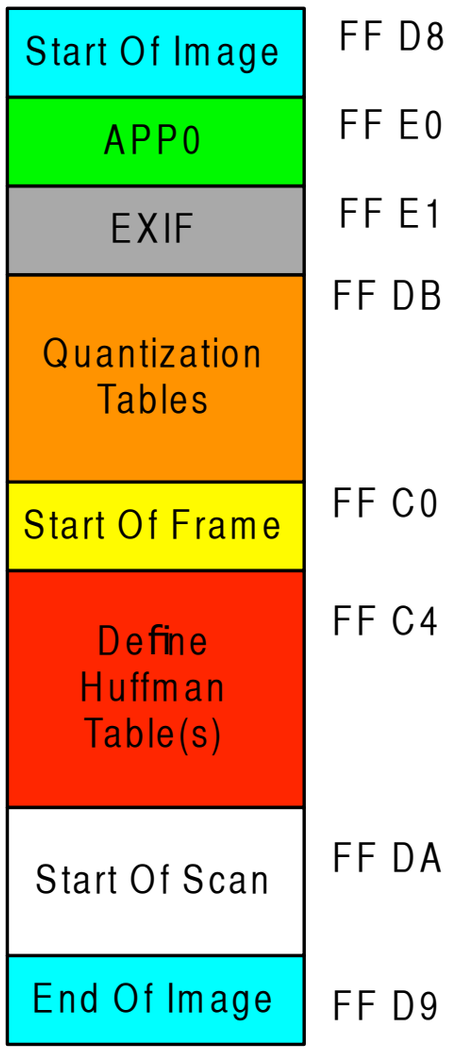
- Data sent between multiple users with multiple devices



These visualizations are typically made by hand.

I use this visualization to *explain* JPEG segments

Omnigraffle



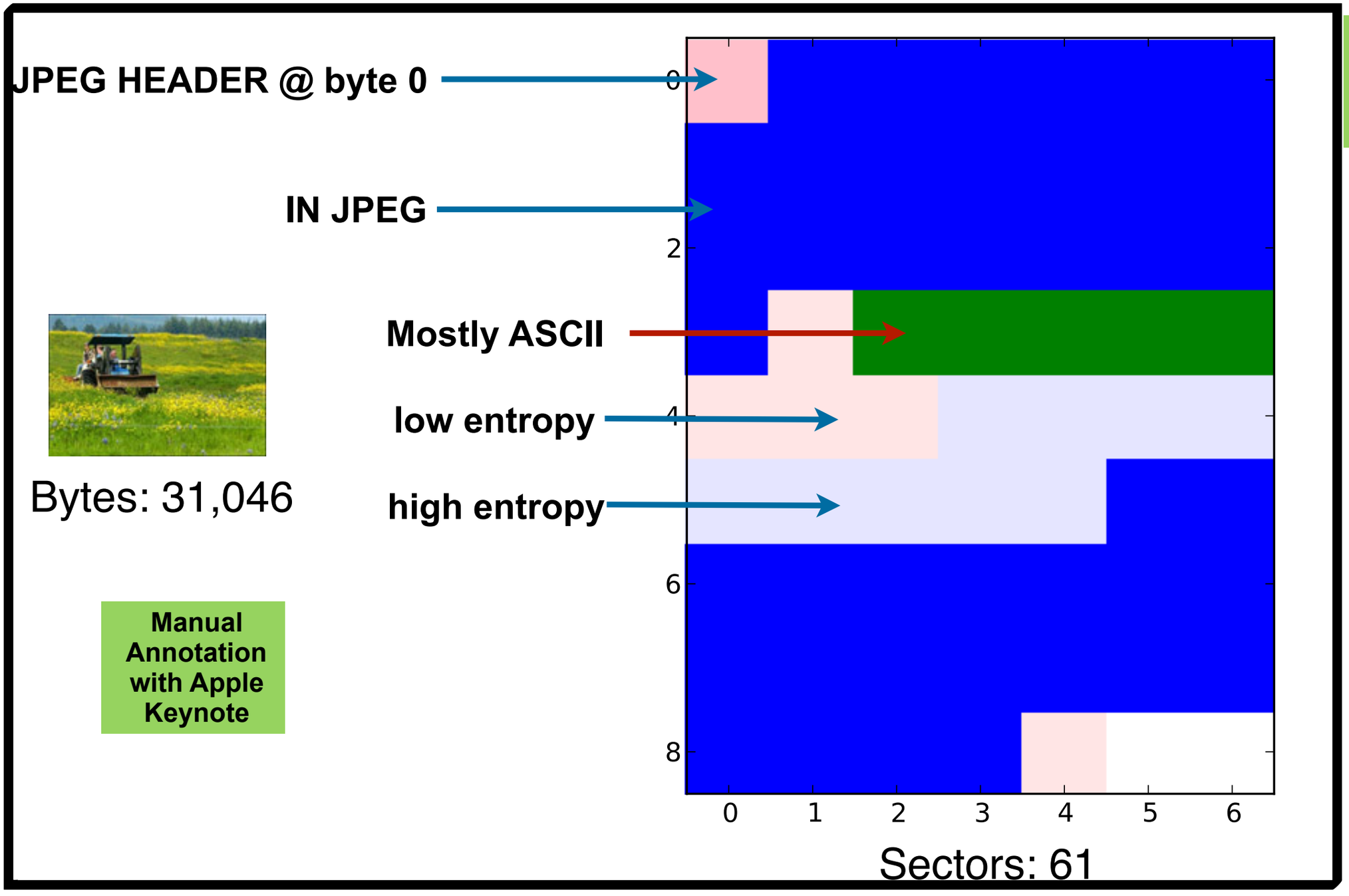
Apple TextEdit

```
$ xxd 1pixel.jpg
0000000: ffd8 ffe0 0010 4a46 4946 0001 0101 0048  ......JFIF.....H
0000010: 0048 0000 ffdb 0043 0003 0202 0302 0203  .H.....C........
0000020: 0303 0304 0303 0405 0805 0504 0405 0a07  ................
0000030: 0706 080c 0a0c 0c0b 0a0b 0b0d 0e12 100d  ................
0000040: 0e11 0e0b 0b10 1610 1113 1415 1515 0c0f  ................
0000050: 1718 1614 1812 1415 14ff c000 0b08 0001  ................
0000060: 0001 0101 1100 ffc4 001f 0000 0105 0101  ................
0000070: 0101 0101 0000 0000 0000 0000 0102 0304  ................
0000080: 0506 0708 090a 0bff c400 b510 0002 0103  ................
0000090: 0302 0403 0505 0404 0000 017d 0102 0300  ...........}....
00000a0: 0411 0512 2131 4106 1351 6107 2271 1432  ....!1A..Qa."q.2
00000b0: 8191 a108 2342 b1c1 1552 d1f0 2433 6272  ....#B...R..$3br
00000c0: 8209 0a16 1718 191a 2526 2728 292a 3435  ........%&'()*45
00000d0: 3637 3839 3a43 4445 4647 4849 4a53 5455  6789:CDEFGHIJSTU
00000e0: 5657 5859 5a63 6465 6667 6869 6a73 7475  VWXYZcdefghijstu
00000f0: 7677 7879 7a83 8485 8687 8889 8a92 9394  vwxyz...........
0000100: 9596 9798 999a a2a3 a4a5 a6a7 a8a9 aab2  ................
0000110: b3b4 b5b6 b7b8 b9ba c2c3 c4c5 c6c7 c8c9  ................
0000120: cad2 d3d4 d5d6 d7d8 d9da e1e2 e3e4 e5e6  ................
0000130: e7e8 e9ea f1f2 f3f4 f5f6 f7f8 f9fa ffda  ................
0000140: 0008 0101 0000 3f00 fd53 afff d9         ......?..S...
```

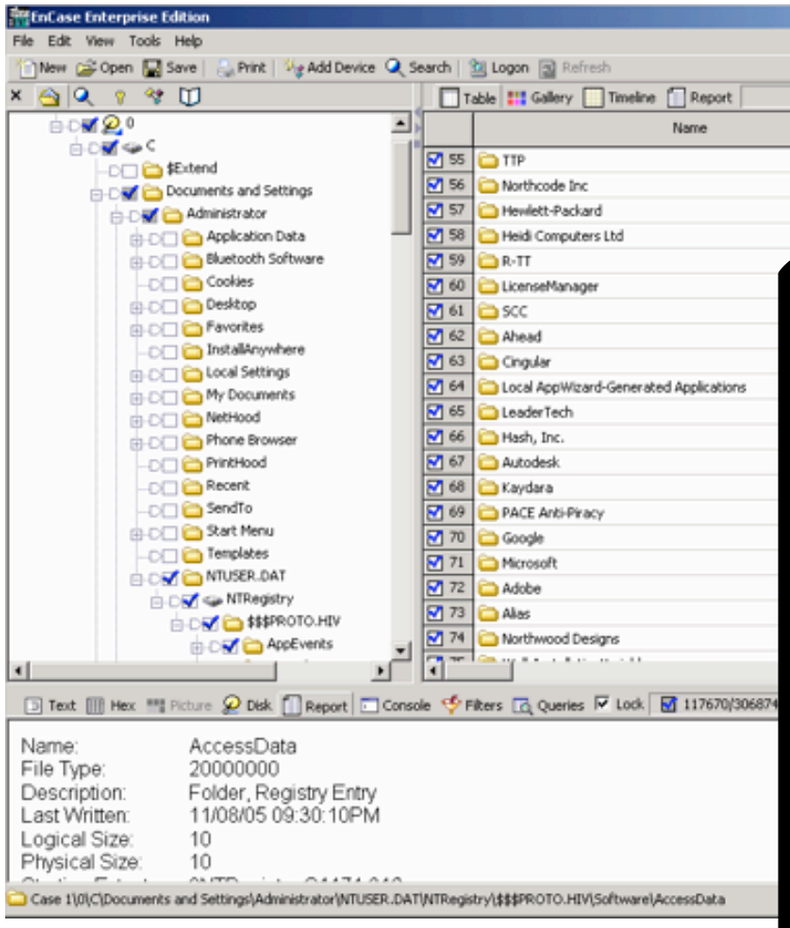
(Inkscape or Matplotlib)

(re-implemented with CSS and HTML)

Explanatory visualizations can have a mix of *data-driven* and *hand-drafted* elements.



Visualizations do not need to be graphical!
Common visualizations are tables, text files, and hex dumps



EnCase Forensic

```
327594963       CPS-requests@verisign.com       m; by E-mail at CPS-requests@verisign.com; or\x0Aby mail at
327734947       CPS-requests@verisign.com       m; by E-mail at CPS-requests@verisign.com; or\x0Aby mail at
327895387       CPS-requests@verisign.com       m; by E-mail at CPS-requests@verisign.com; or\x0Aby mail at
328057251       CPS-requests@verisign.com       m; by E-mail at CPS-requests@verisign.com; or\x0Aby mail at
332652110       CPS-requests@verisign.com       m; by E-mail at\x0ACPS-requests@verisign.com; or by mail at
332653035       CPS-requests@verisign.com       0; by E-mail at CPS-requests@verisign.com; or by mail at
332653461       CPS-requests@verisign.com       S; by E-mail at CPS-requests@verisign.com; or by mail at
370161659       intranet2000@banamex.com        ea href="mailto:intranet2000@banamex.com" shape="rect" c
345020131       CPS-requests@verisign.com       m; by E-mail at CPS-requests@verisign.com; or\x0Aby mail at
347139766       CPS-requests@verisign.com       m; by E-mail at CPS-requests@verisign.com; or\x0Aby mail at
```

bulk_extractor feature file

- Easy:

```
0000000: ffd8 ffe0 0010 4a46 4946 0001 0201 0048  ......JFIF.....H
0000010: 0048 0000 ffe1 1d17 4578 6966 0000 4d4d  .H......Exif..MM
0000020: 002a 0000 0008 0007 0112 0003 0000 0001  .*..............
0000030: 0001 0000 011a 0005 0000 0001 0000 0062  ...............b
0000040: 011b 0005 0000 0001 0000 006a 0128 0003  ...........j.(..
0000050: 0000 0001 0002 0000 0131 0002 0000 001b  .........1......
0000060: 0000 0072 0132 0002 0000 0014 0000 008d  ...r.2..........
0000070: 8769 0004 0000 0001 0000 00a4 0000 00d0  .i..............
0000080: 0000 0048 0000 0001 0000 0048 0000 0001  ...H.......H....
0000090: 4164 6f62 6520 5068 6f74 6f73 686f 7020  Adobe Photoshop
00000a0: 4353 2057 696e 646f 7773 0032 3030 353a  CS Windows.2005:
00000b0: 3035 3a30 3920 3136 3a30 313a 3432 0000  05:09 16:01:42..
00000c0: 0000 0003 a001 0003 0000 0001 0001 0000  ................
00000d0: a002 0004 0000 0001 0000 00c8 a003 0004  ................
00000e0: 0000 0001 0000 0084 0000 0000 0000 0006  ................
00000f0: 0103 0003 0000 0001 0006 0000 011a 0005  ................
```

- Hard:

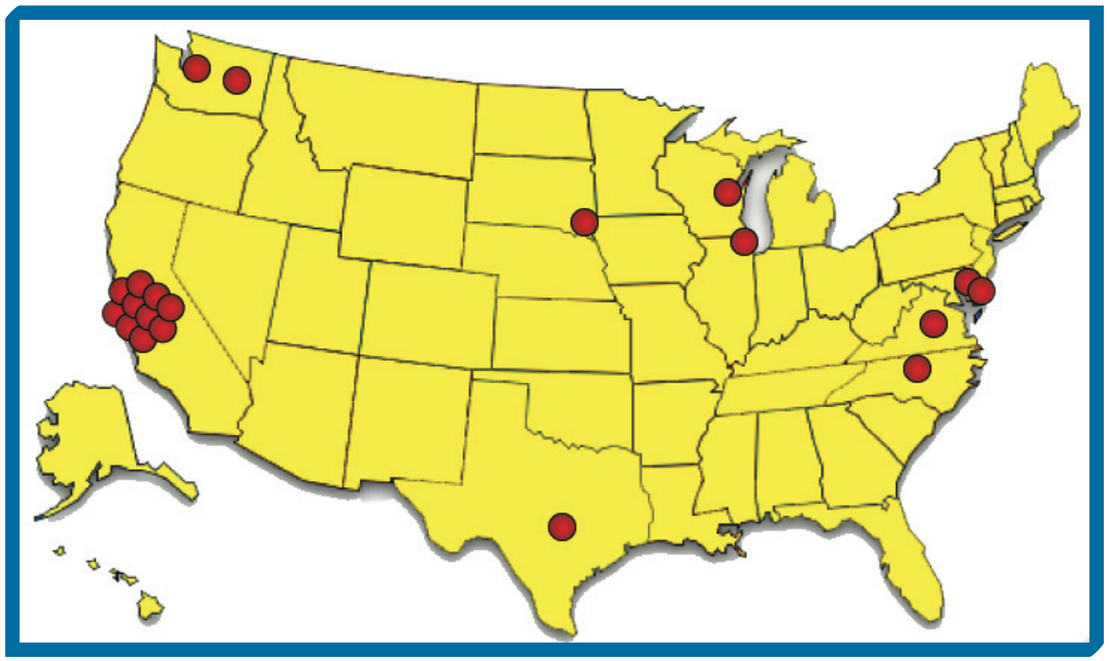
```
000a000: 0011 fa71 57f4 6f5f ddff 00bd 15fb 5dfd  ...qW.o_......].
000a010: a996 0fc9 dff1 ff00 b149 e154 97f4 efd5  .........I.T....
000a020: e3f5 7f47 71df 8ffb d5d7 da9e d87f c12f  ...Gq........../
000a030: f8ff 00d8 b1f4 b1f8 ff00 c57e ab7a ff00  ..........~.z..
```

hex dump

Great for discovery... but not very exciting

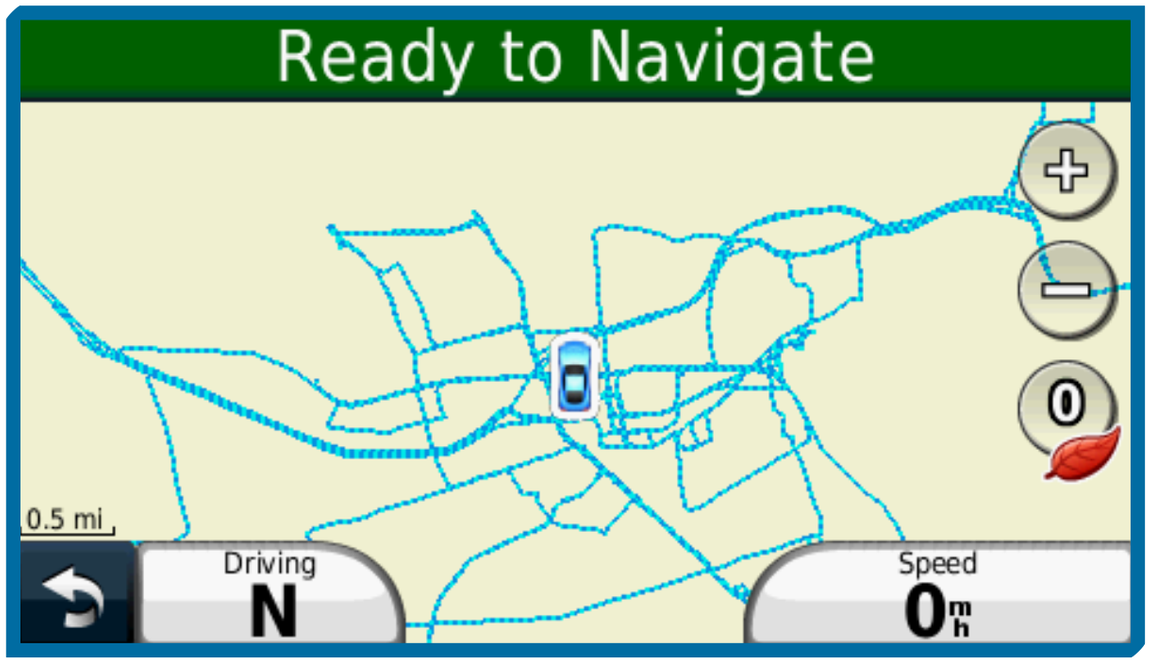
Instant situational awareness—fast and easy to interpret.

Graphical visualizations are a great way to show geospatial information.



Garfinkel 2005
Omnigraffle

"Hand-drafted"



Garmin GPS

"Data-driven"

Data-driven visualizations are great for showing graphs.

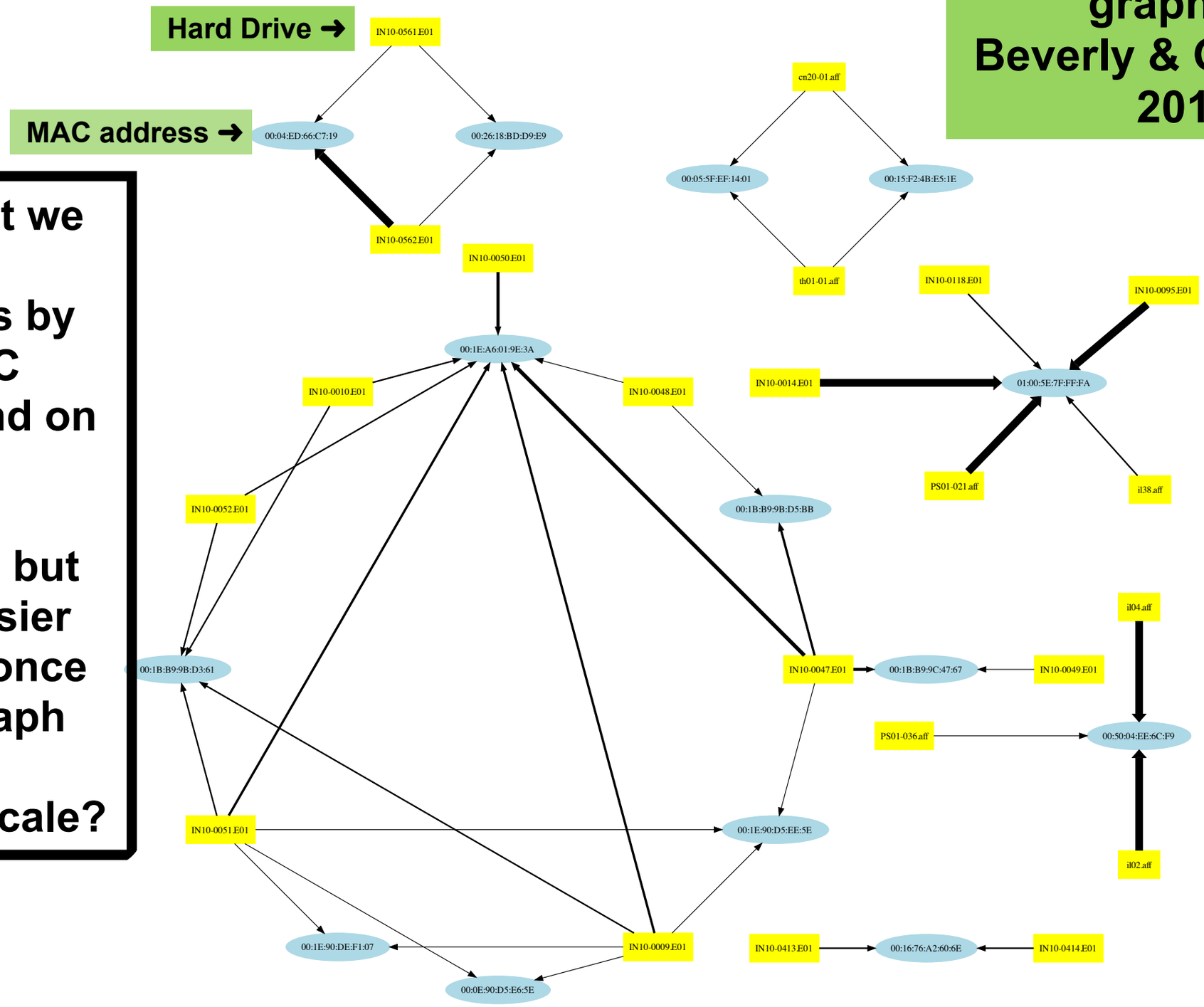
Here we used the graph to test a hypothesis

bulk_extractor + graphiviz Beverly & Garfinkel, 2010

We thought that we could identify groups of users by correlating MAC addresses found on hard drives

We had a table, but it was much easier to understand once we drew the graph

But does this scale?



We can build data-driven visualizations with JavaScript kits. e.g. http://d3js.org/

Methodology:

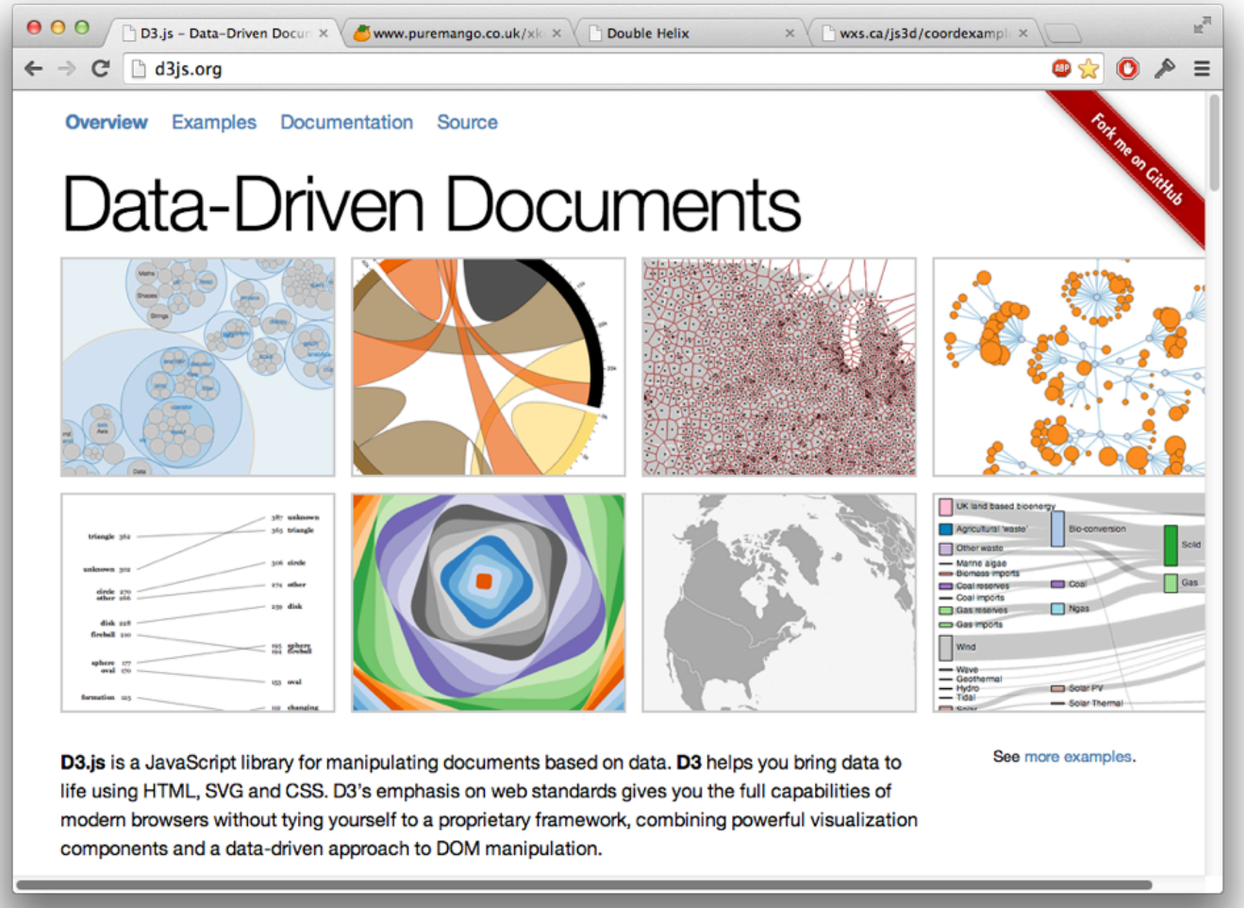
- Create HTML body, CSS style & JSON data model
- JavaScript reads JSON and creates SVG data elements
- Layout engine makes everything look good

Advantages:

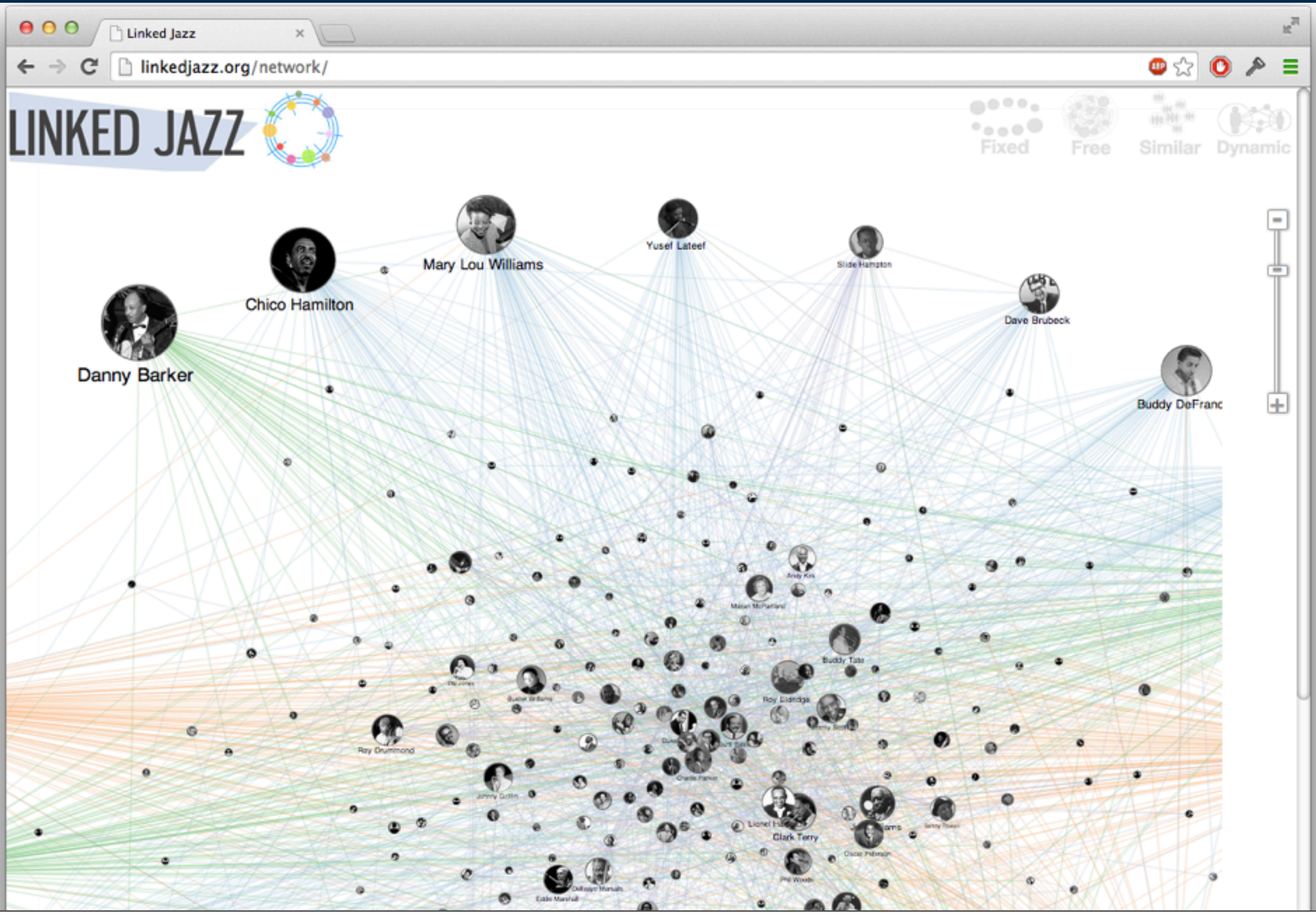
- Everybody has a browser
- Browsers do layout, fonts, etc.
- CSS offers a lot of flexibility
- JavaScript engines can handle big datasets

Issues:

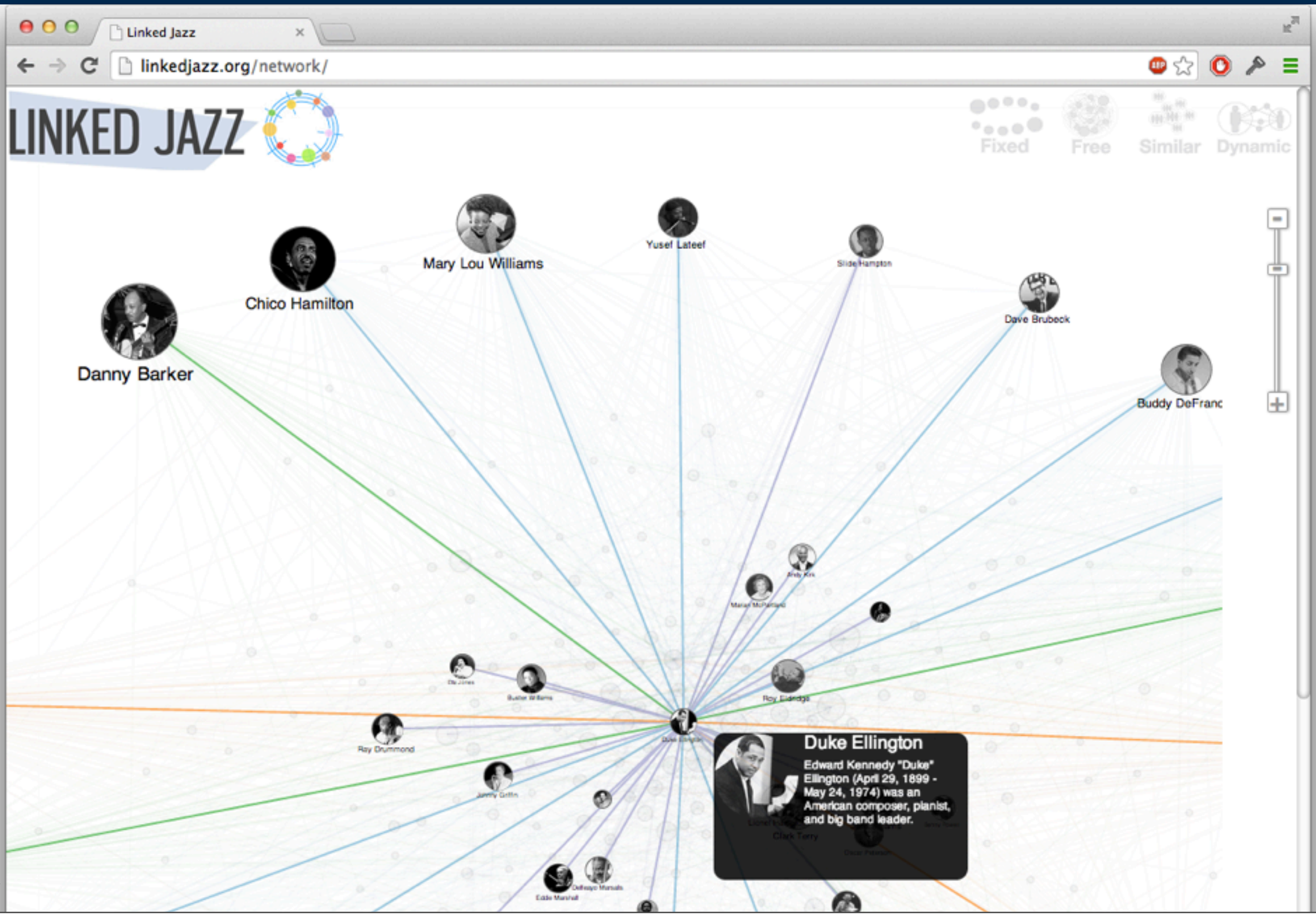
- Requires HTML & CSS design
- Heavy use of JQuery
- Must “fix” output for forensic use



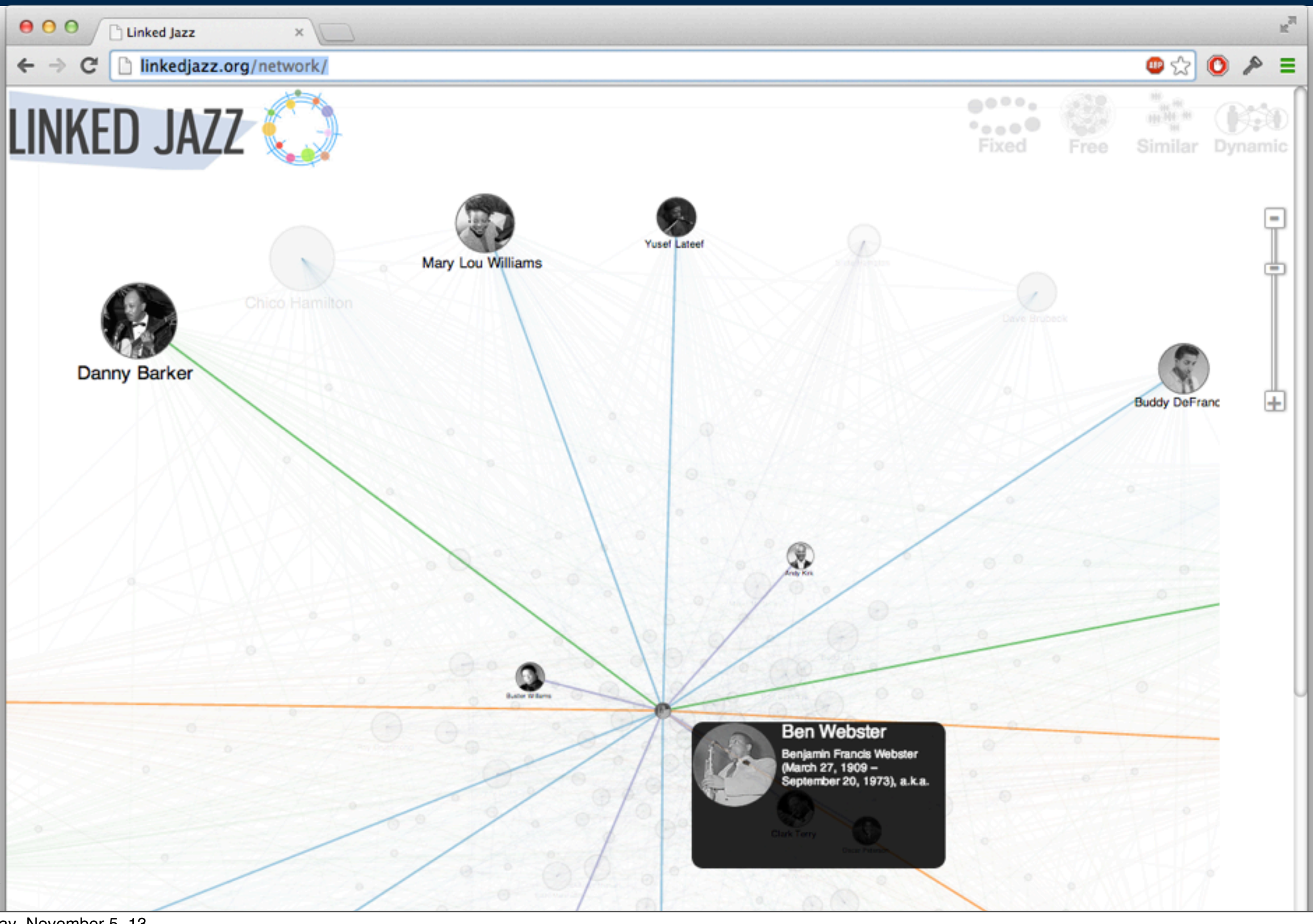
Data-Driven documents should allow for discovery! (e.g. http://linkedjazz.org/network/)



Many of these documents rely on interactivity.

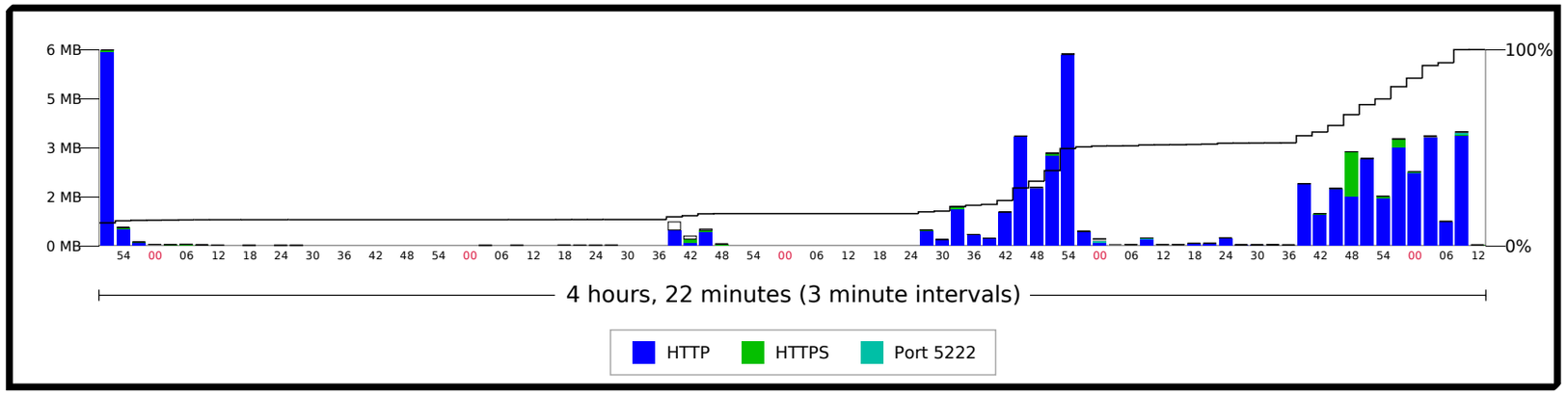


But if you move the mouse, you get a different result.

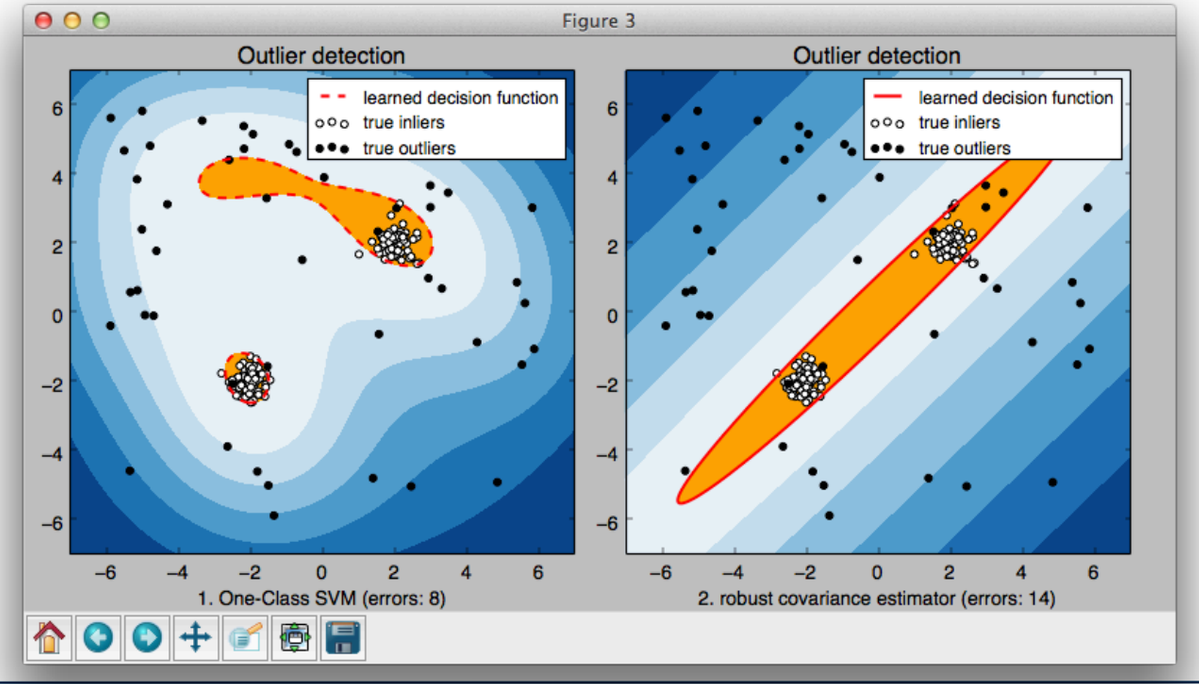


We are trying to create data-driven visualizations for forensic discovery.

Spot data / trends that were not obvious



Detect clusters & outliers



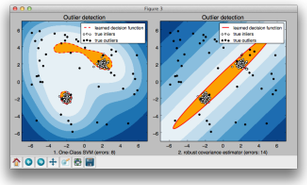
Visualizations for forensic discovery should be “automatic.”

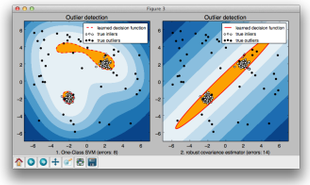
Data driven

- You create the code — ONCE
- Different data ➜ different graphics

Fixed, predictable, static output:

- Interactive visualizations aren’t appropriate for court
- Different analysts should produce the same visualization



What about interactivity?

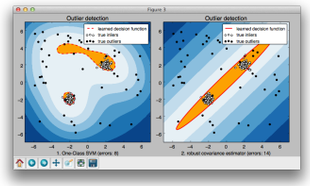
- Use interactivity for finding **settings**
- Use the **settings** for producing the visualization

What about video?

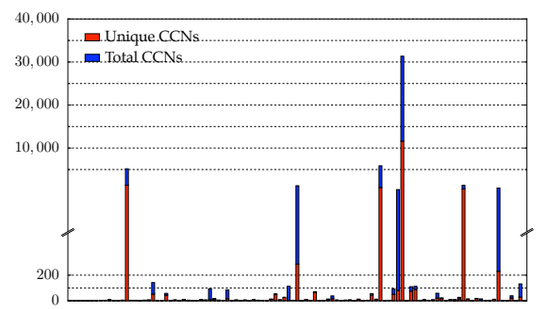
- Video doesn’t work well in most reports. Is it needed?

This presentation is about making static visualizations of computer forensic data with open source tools.

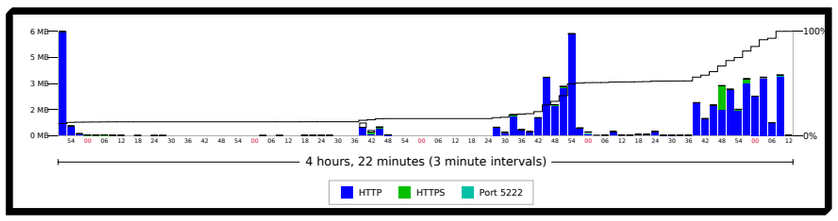
Why we want static visualization



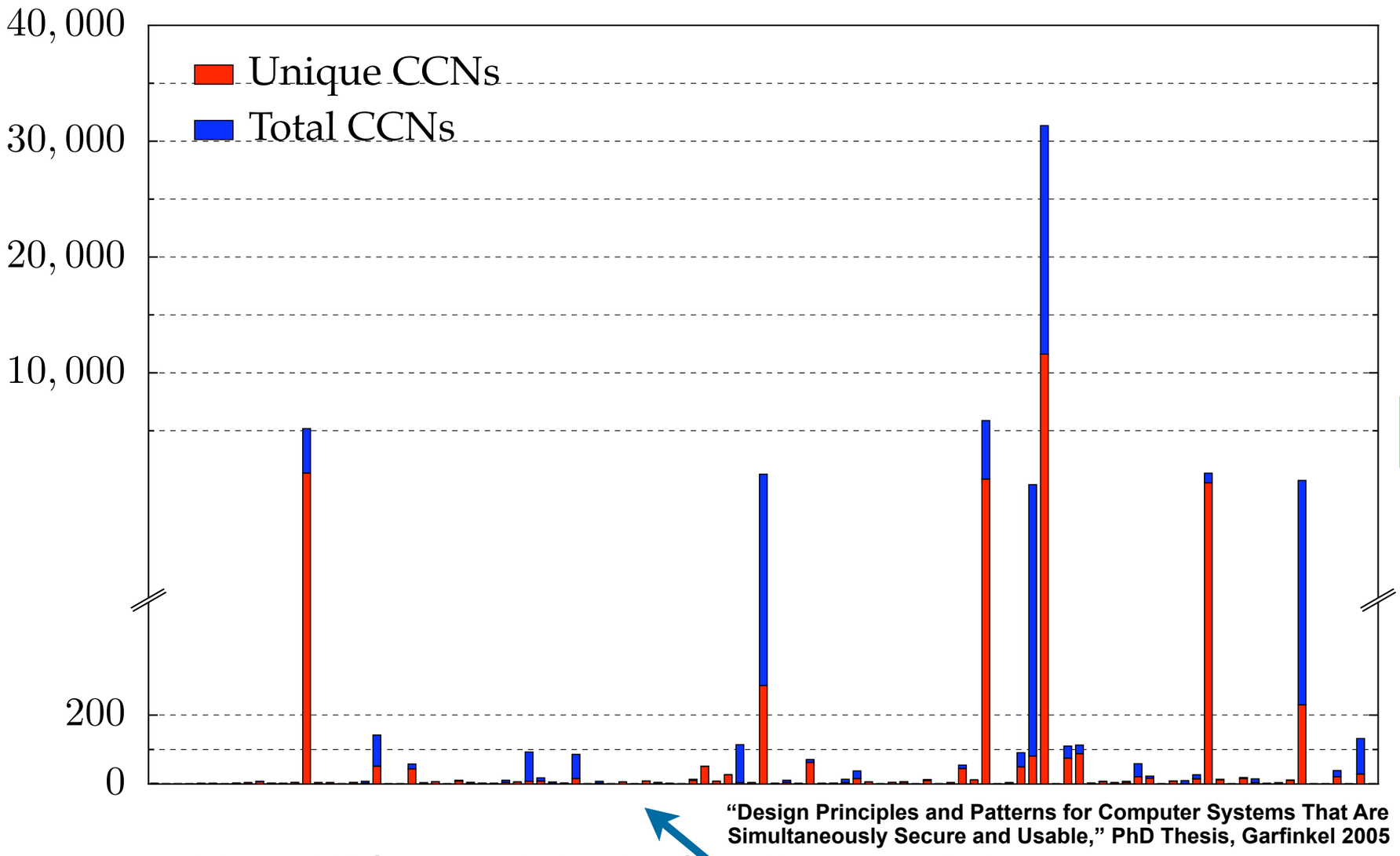
Issues to consider when making visualizations



How we created a visualization for tcpflow



This data-driven graph shows incidence of credit card numbers on a collection of hard drives.



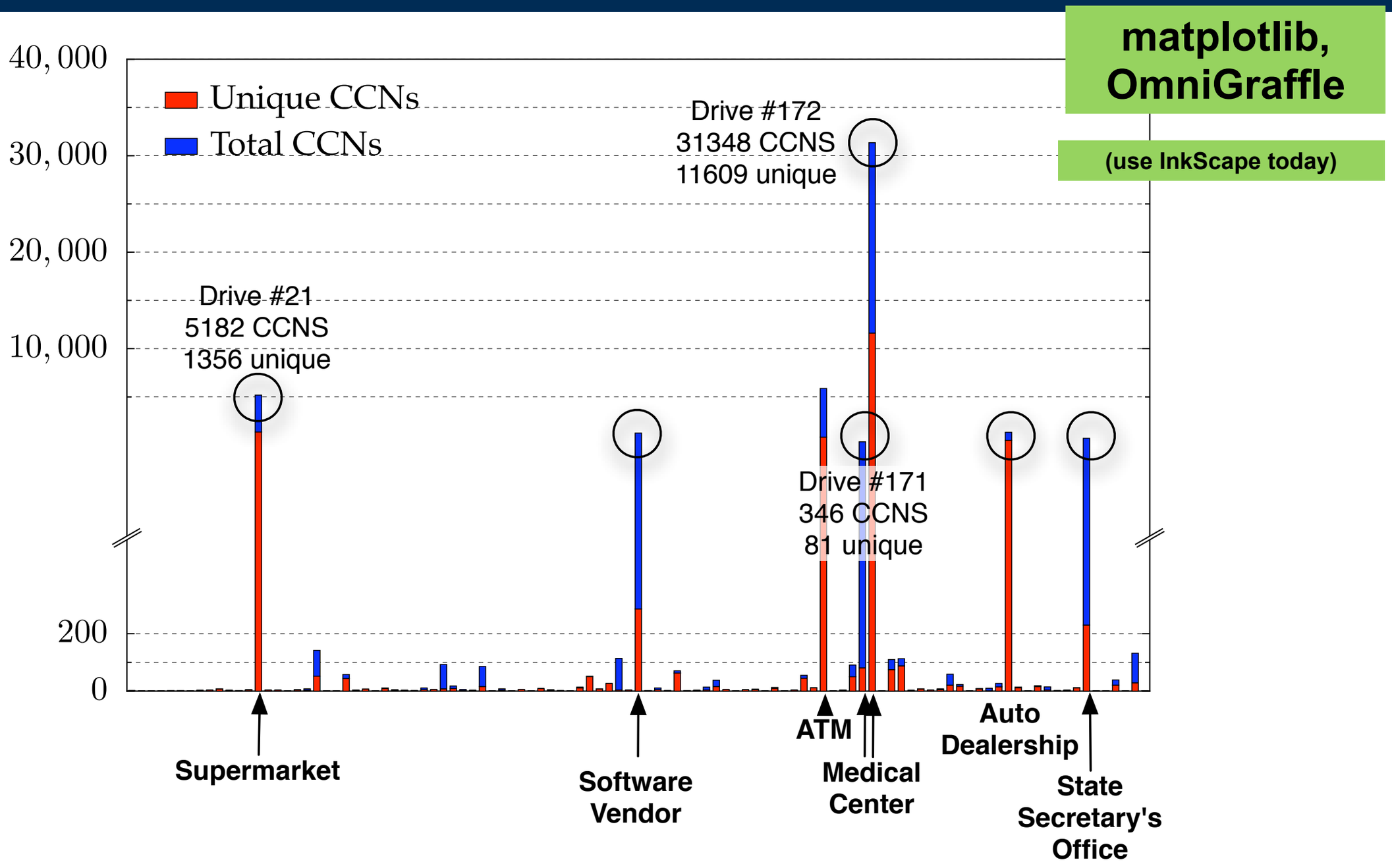
—*Data source: strings(1) | ccn_detector | python_scripts*

The graph demonstrates:

- outliers
- total vs. unique

X axis is time of acquisition
It's not meaningful...

The same graph, annotated based on interviews.



- Data Sources: (previous PDF) + phone calls

Today this is an easier graph to make.

Steps:

1. Run bulk_extractor on disk drives
2. Read the (# unique) and (# total) CCNs in the feature & histogram file
3. Create a bar graph with matplotlib

We ran bulk_extractor on a few thousand drives.

Results were stored in .zip files:

```
/corp/nus/drives_bulk_extractor-2013-08-20:
total used in directory 90286552 available 5713959760
drwxr-xr-x.  2 simsong root        61440 Aug 20 14:41 .
drwxr-x---. 14 corpus  irb          4096 Sep 30 09:03 ..
-rw-rw-r--.  1 simsong simsong     31513 Jul 30 11:07 0305.zip
-rw-rw-r--.  1 simsong simsong   1835403 Jul 30 11:07 0308.zip
-rw-rw-r--.  1 simsong simsong    833418 Jul 30 11:07 0310.zip
-rw-rw-r--.  1 simsong simsong     20005 Jul 30 11:07 0312.zip
-rw-rw-r--.  1 simsong simsong  15921378 Jul 30 11:07 0313.zip
-rw-rw-r--.  1 simsong simsong   2938098 Jul 30 11:07 0314.zip
-rw-rw-r--.  1 simsong simsong   3547615 Jul 30 11:07 0315.zip
-rw-rw-r--.  1 simsong simsong     16468 Jul 30 11:07 0498.zip
-rw-rw-r--.  1 simsong simsong     11287 Jul 30 11:06 0572.zip
-rw-rw-r--.  1 simsong simsong     11285 Jul 30 11:07 0574.zip
```

Each ZIP file contains carved objects and feature files:

Length	Date	Time	File
0	28-Jul-2013	10:12:10	0313/
3064	28-Jul-2013	10:09:06	0313/ccn.txt
435	28-Jul-2013	10:12:02	0313/ccn_histogram.txt
19440204	28-Jul-2013	10:11:36	0313/domain.txt
47932	28-Jul-2013	10:12:02	0313/email_histogram.txt
2291	28-Jul-2013	10:01:44	0313/ether.txt
0	28-Jul-2013	08:50:02	0313/find.txt
1675013	28-Jul-2013	10:11:22	0313/jpeg.txt
0	28-Jul-2013	10:06:24	0313/jpeg/
27702	28-Jul-2013	10:01:06	0313/jpeg/17849683887-ZIP-0.jpg
18031	28-Jul-2013	10:01:06	0313/jpeg/17849727138-ZIP-0.jpg
13734	28-Jul-2013	10:01:12	0313/jpeg/17853205179-ZIP-0.jpg
7393	28-Jul-2013	10:01:12	0313/jpeg/17853264313-ZIP-0.jpg
7805	28-Jul-2013	10:01:12	0313/jpeg/17853270606-ZIP-0.jpg
8799	28-Jul-2013	10:01:12	0313/jpeg/17853290623-ZIP-0.jpg
9358	28-Jul-2013	10:01:12	0313/jpeg/17853298338-ZIP-0.jpg
7446	28-Jul-2013	10:01:12	0313/jpeg/17855329427-ZIP-0.jpg
7267	28-Jul-2013	10:01:12	0313/jpeg/17855335786-ZIP-0.jpg
8407	28-Jul-2013	10:01:12	0313/jpeg/17855341964-ZIP-0.jpg
8951	28-Jul-2013	10:01:12	0313/jpeg/17855349323-ZIP-0.jpg
9861	28-Jul-2013	10:01:12	0313/jpeg/17855366469-ZIP-0.jpg
10079	28-Jul-2013	10:06:24	0313/jpeg/19117936375-ZIP-0.jpg
18488	28-Jul-2013	10:10:48	0313/json.txt
1155	28-Jul-2013	10:06:46	0313/rar.txt
11166	28-Jul-2013	10:12:02	0313/telephone_histogram.txt
0	28-Jul-2013	10:12:08	0313/url_facebook-address.txt
27521	28-Jul-2013	10:12:10	0313/url_searches.txt
230917	28-Jul-2013	10:12:08	0313/url_services.txt
0	28-Jul-2013	08:50:02	0313/vcard.txt
11047565	28-Jul-2013	10:11:40	0313/windirs.txt

lines in feature file = # of total CCNs
lines in histogram file = # of “distinct” CCNs

Feature file:

```
..
15759011793     487462553450448         2;sz=120x60;ord=487462553450448\x00\x0D
15759013329     487462553450448         3;sz=120x60;ord=487462553450448\x00\x0D
15768826985     476347676736456         7654'6767676767'476347676736456?\x01676
..
```

These are not real CCNs (false positives)

Histogram file:

```
n=10    487462553450448
n=3     476347676736456
n=3     567432627676763 2
n=2     422866533000444 9
n=2     626103114233300 0
n=2     657738324738546 1
n=1     435324524635635 2
n=1     498054165276498 5
n=1     556666778888999
n=1     644433356523352 1
n=1     SSN: 666-66-6666
```

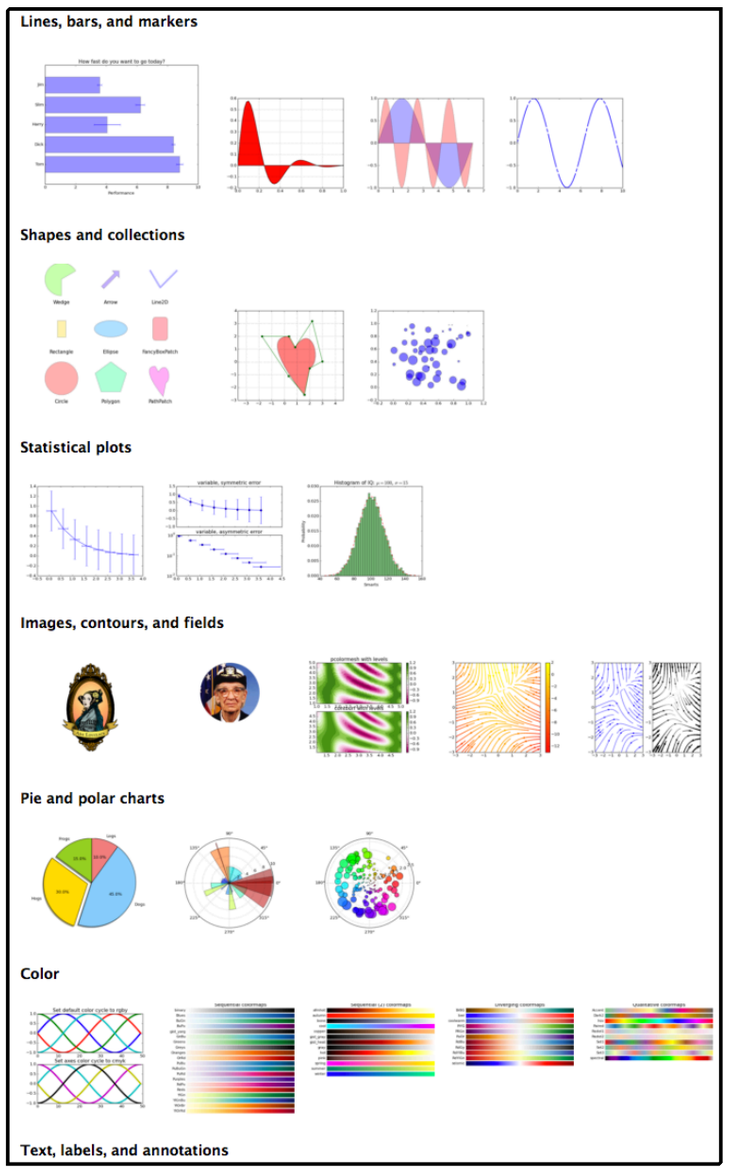
We store this in an SQL database:

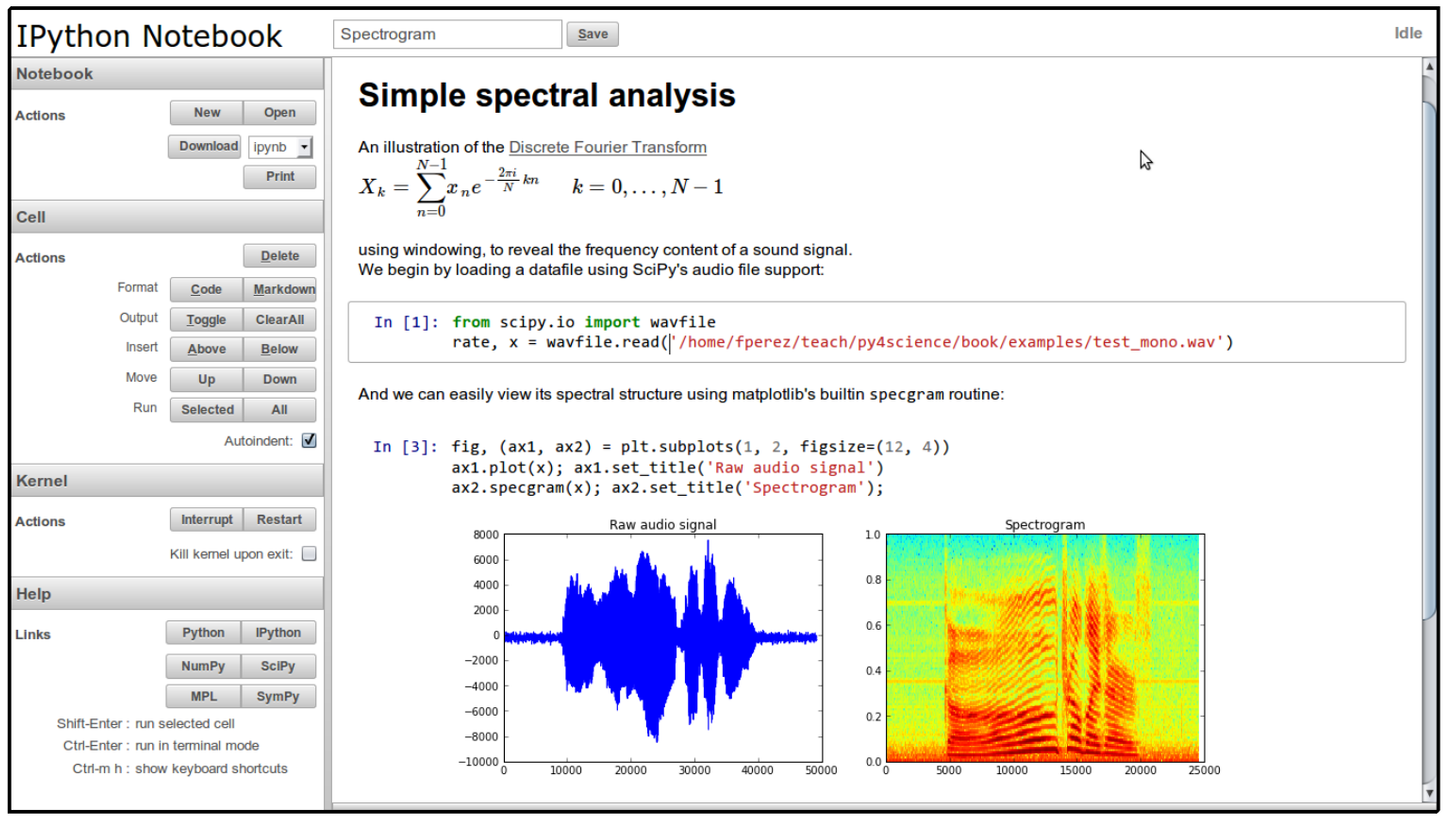
```
CREATE TABLE files (fileid INTEGER PRIMARY KEY ASC,report_filename TEXT
UNIQUE,image_filename TEXT UNIQUE);
CREATE TABLE features (featureid INTEGER PRIMARY KEY ASC,featurename TEXT UNIQUE);
CREATE TABLE counts (countid INTEGER PRIMARY KEY ASC,fileid INTEGER,
                     featureid INTEGER,count INTEGER,
        FOREIGN KEY (fileid) references files(fileid),
        FOREIGN KEY (featureid) references features(featureid));
CREATE UNIQUE INDEX counts_idx ON counts (fileid,featureid);
```

It’s easier to work with data in a database

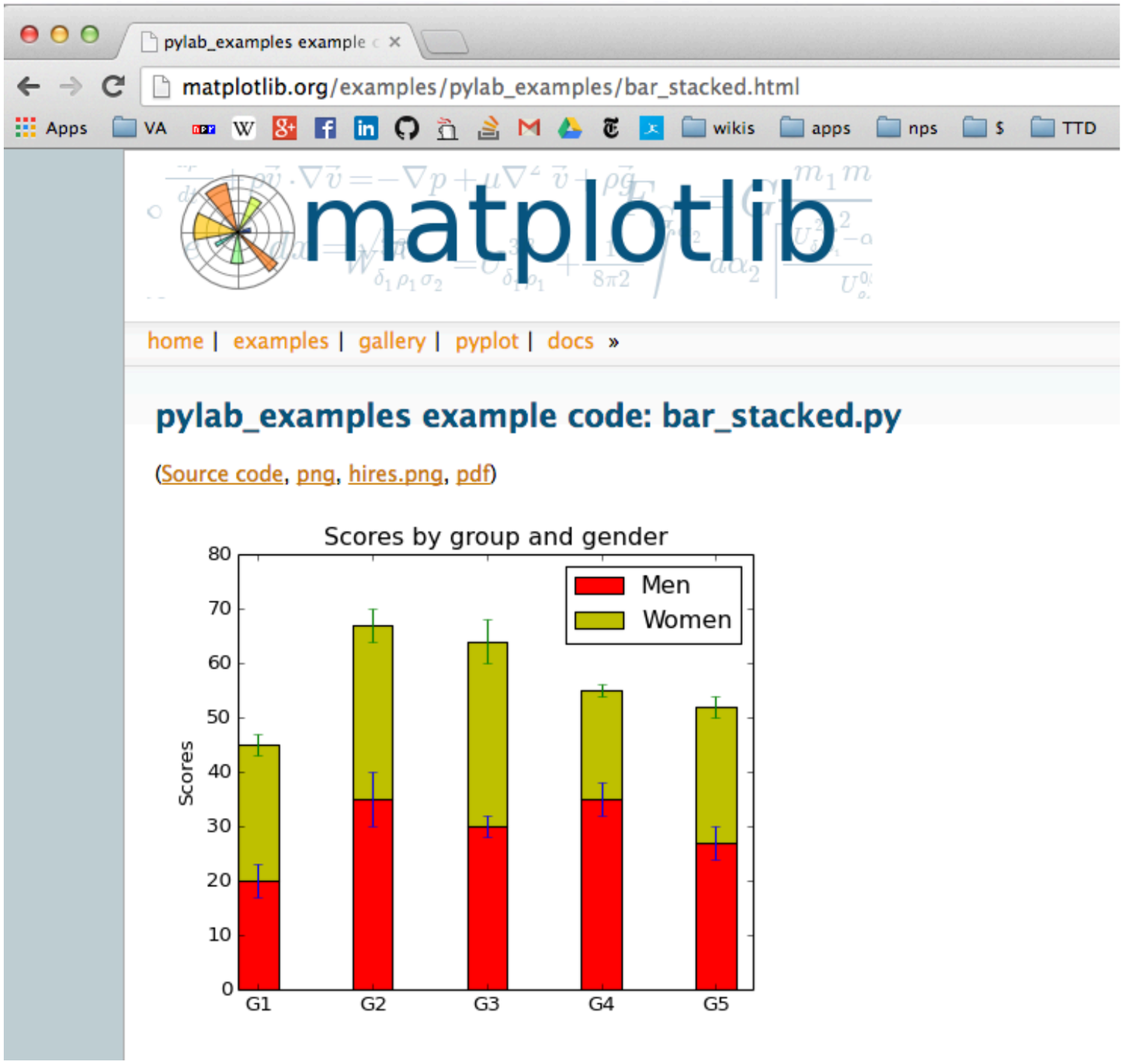
matplotlib is a python library for making visualizations.

- Python 2 & 3 support
- Multiple output formats
- Integrates with pylab and IPython Notebook





Start by looking at the matplotlib gallery for a similar graph.

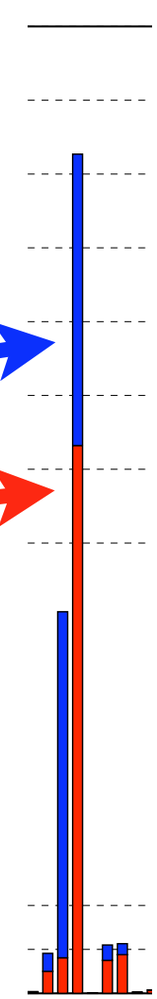


With the data in a DB, extracting the data we want is easy.

```
select image_filename,featurename,count from counts
   natural left join files
   natural left join features
   where featurename in ('ccn.txt','ccn_histogram.txt')
   order by 1;
```

- Produces:

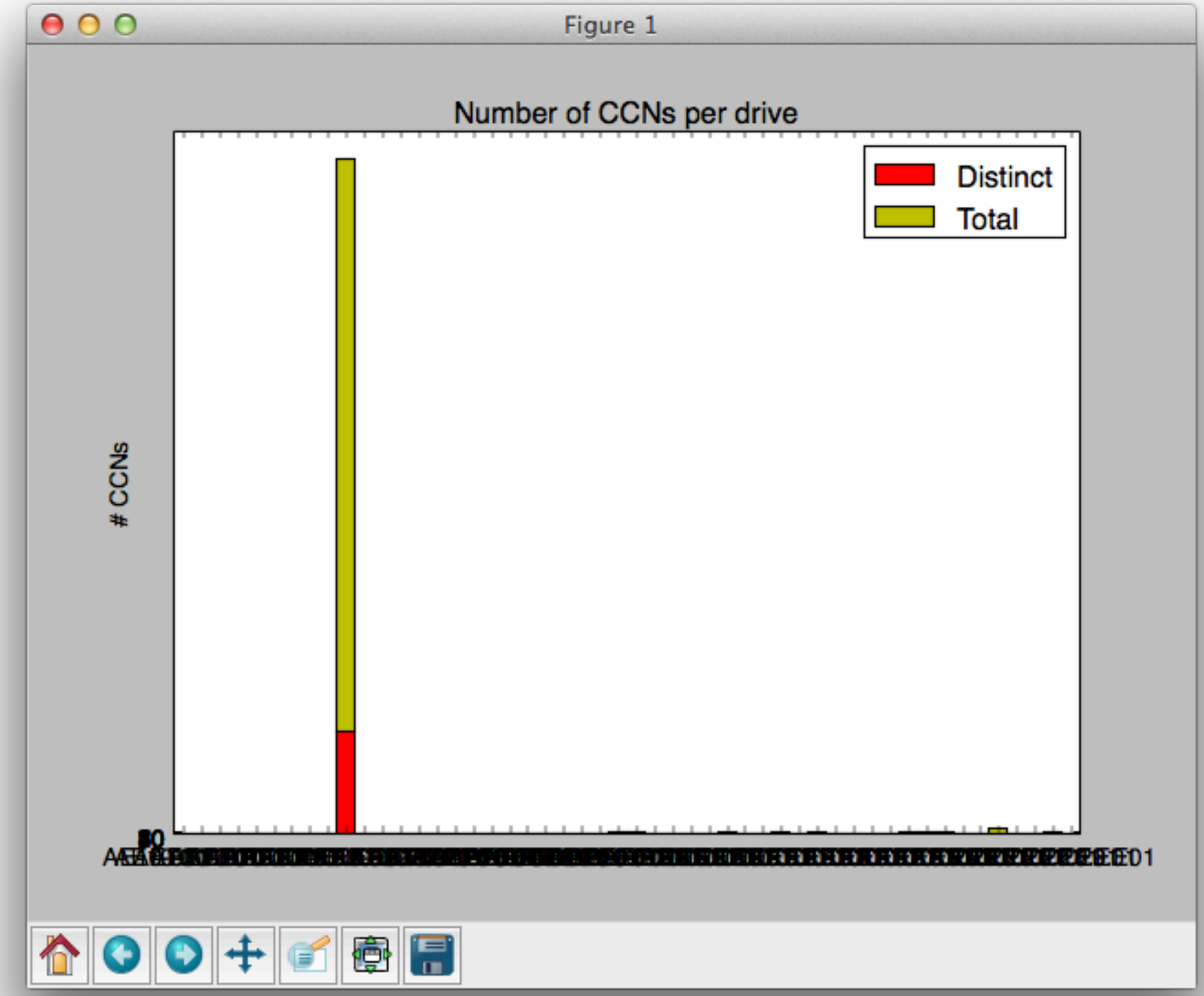
```
AE10-001.E01|ccn_histogram.txt|0
AE10-0010.E01|ccn.txt|93
AE10-0010.E01|ccn_histogram.txt|18
AE10-0011.E01|ccn_histogram.txt|0
AE10-0012.E01|ccn_histogram.txt|0
AE10-0013.E01|ccn.txt|3
AE10-0013.E01|ccn_histogram.txt|1
AE10-0014.E01|ccn_histogram.txt|0
```



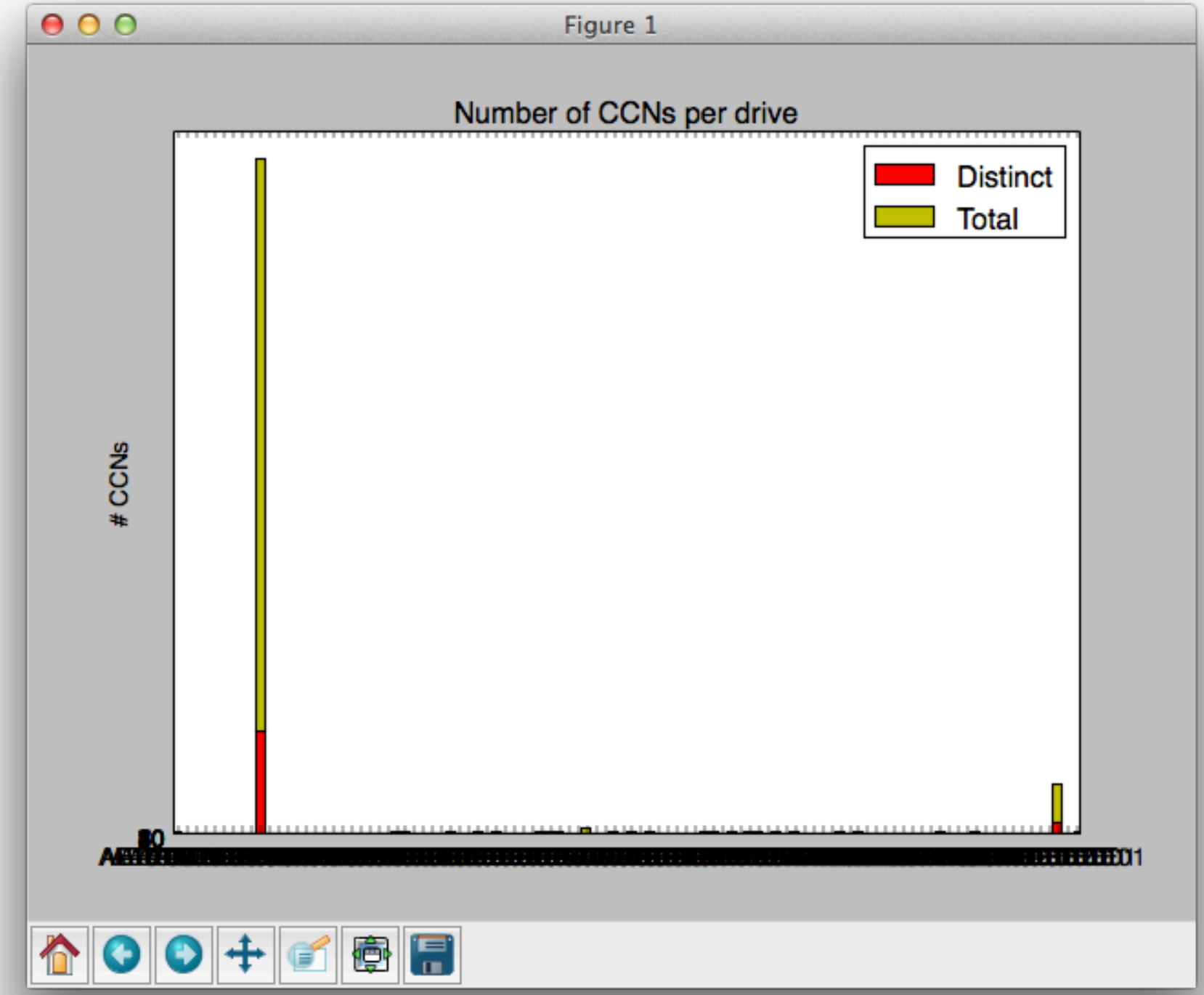
Plot from python using matplotlib

```
def ccngraph(count):
    import numpy as np
    import matplotlib.pyplot as plt

    c = conn.cursor()
    c.execute("select image_filename,count from counts natural left join files natural left join
features where featurename='ccn.txt'");
    totals = dict(c.fetchall())

    c.execute("select image_filename,count from counts natural left join files natural left join
features where featurename='ccn_histogram.txt'");
    distinct = dict(c.fetchall())

    keys = sorted(list(set(list(totals.keys()) + list(distinct.keys()))))
    names = []
    distinctCounts = []
    totalCounts = []

    for k in keys[0:count]:
        names          += [os.path.basename(k)]
        distinctCounts += [distinct.get(k,0)]
        totalCounts    += [totals.get(k,0)]

    ind = np.arange(count)    # the x locations for the groups
    width = 1.0               # the width of the bars: can also be len(x) sequence

    p1 = plt.bar(ind, distinctCounts,   width, color='r')
    p2 = plt.bar(ind, totalCounts, width, color='y',bottom=distinctCounts)

    plt.ylabel('# CCNs')
    plt.title('Number of CCNs per drive')
    plt.xticks(ind+width/2., names )
    plt.yticks(np.arange(0,81,10))
    plt.legend( (p1[0], p2[0]), ('Distinct', 'Total') )
    plt.show()
```

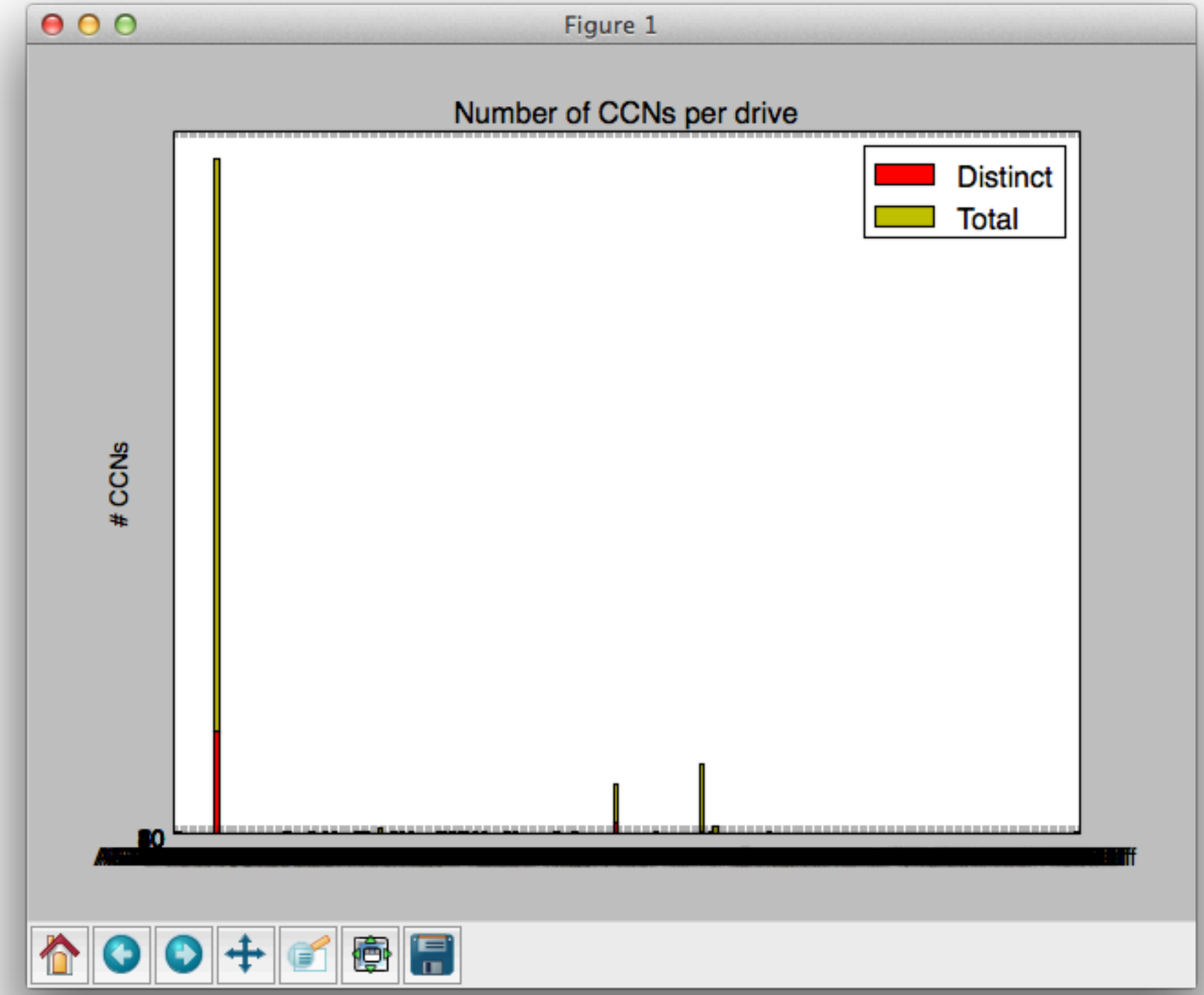
The result is one HUGE bar and lots of little ones. (First 50)



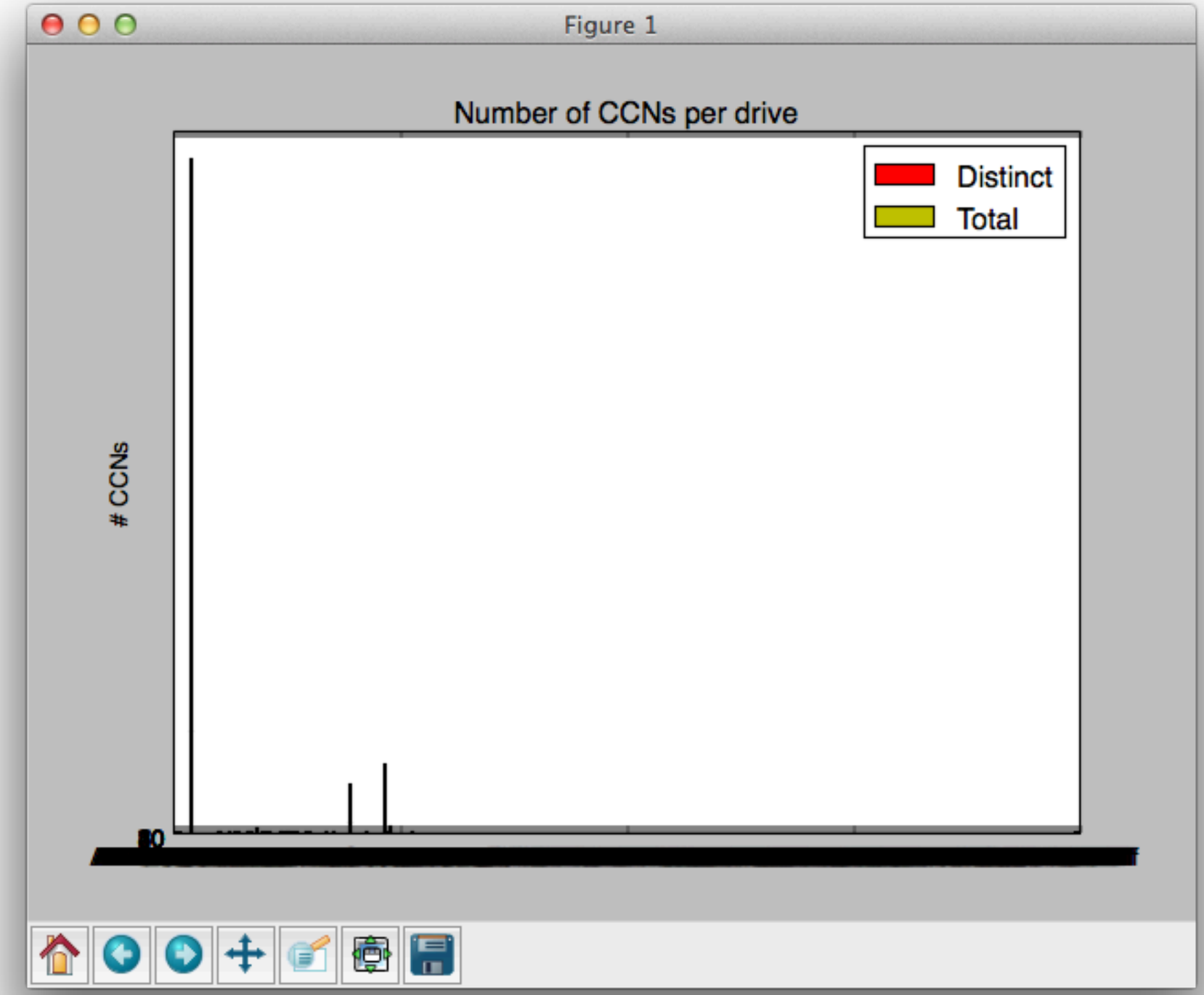
Viewing first 100, there is another large bar.



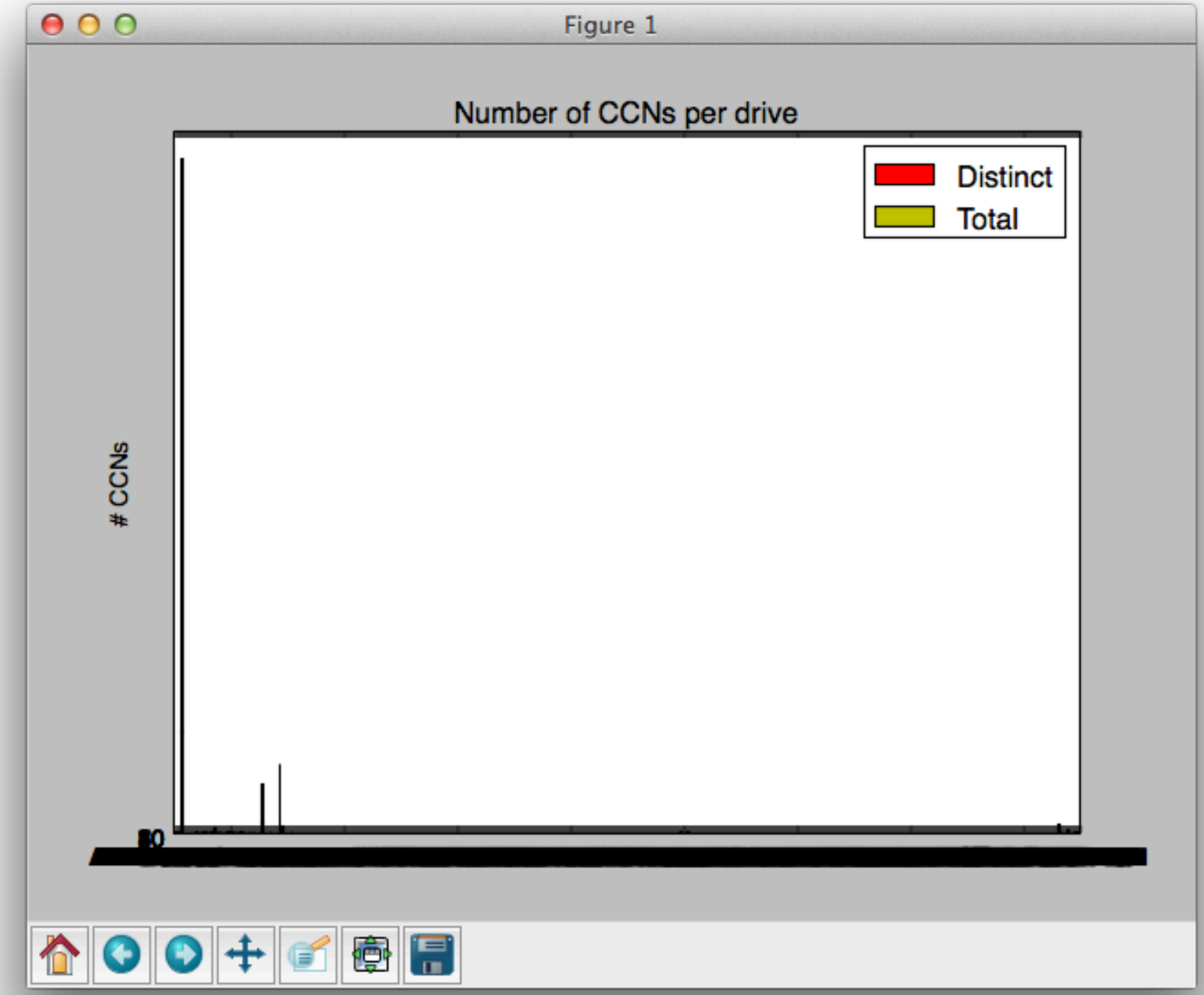
First 200...



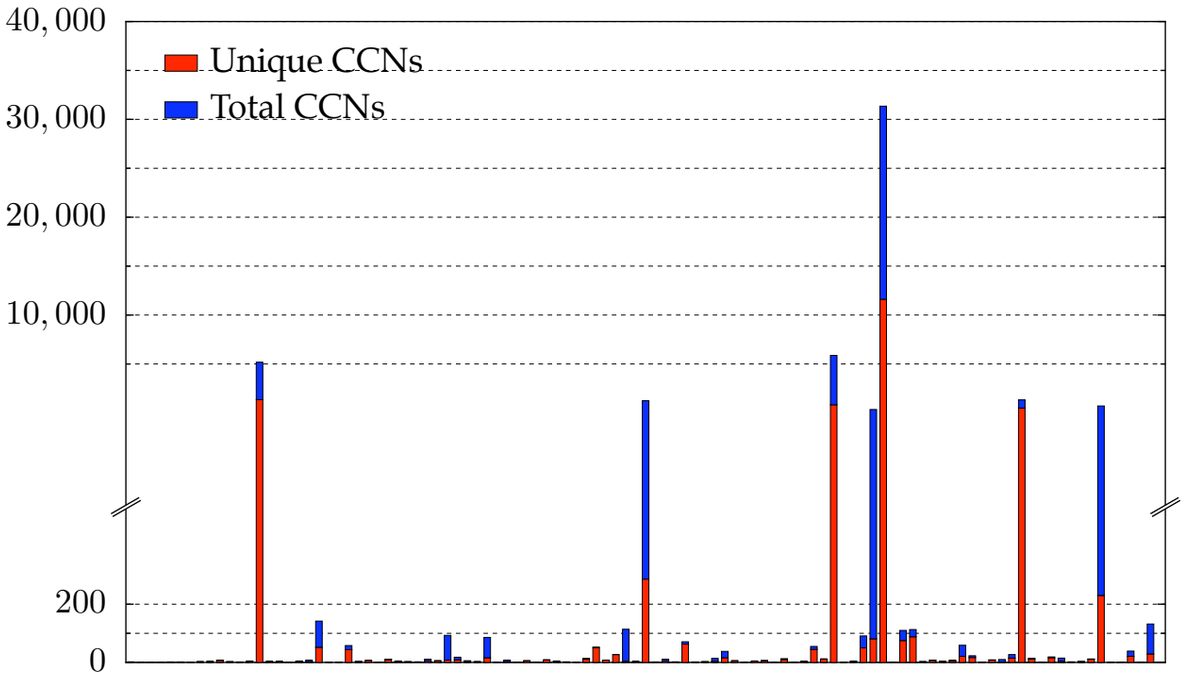
First 500...

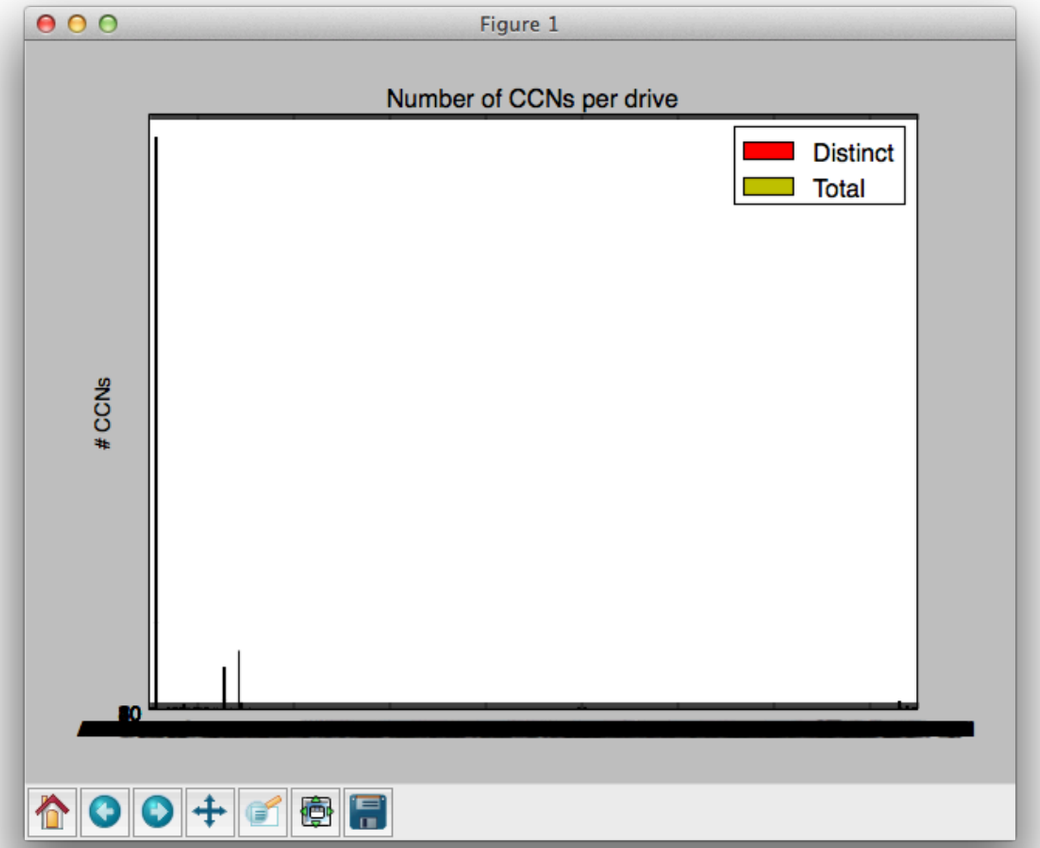


First 1000...



Today's graph doesn't look as good... why?





Key differences:

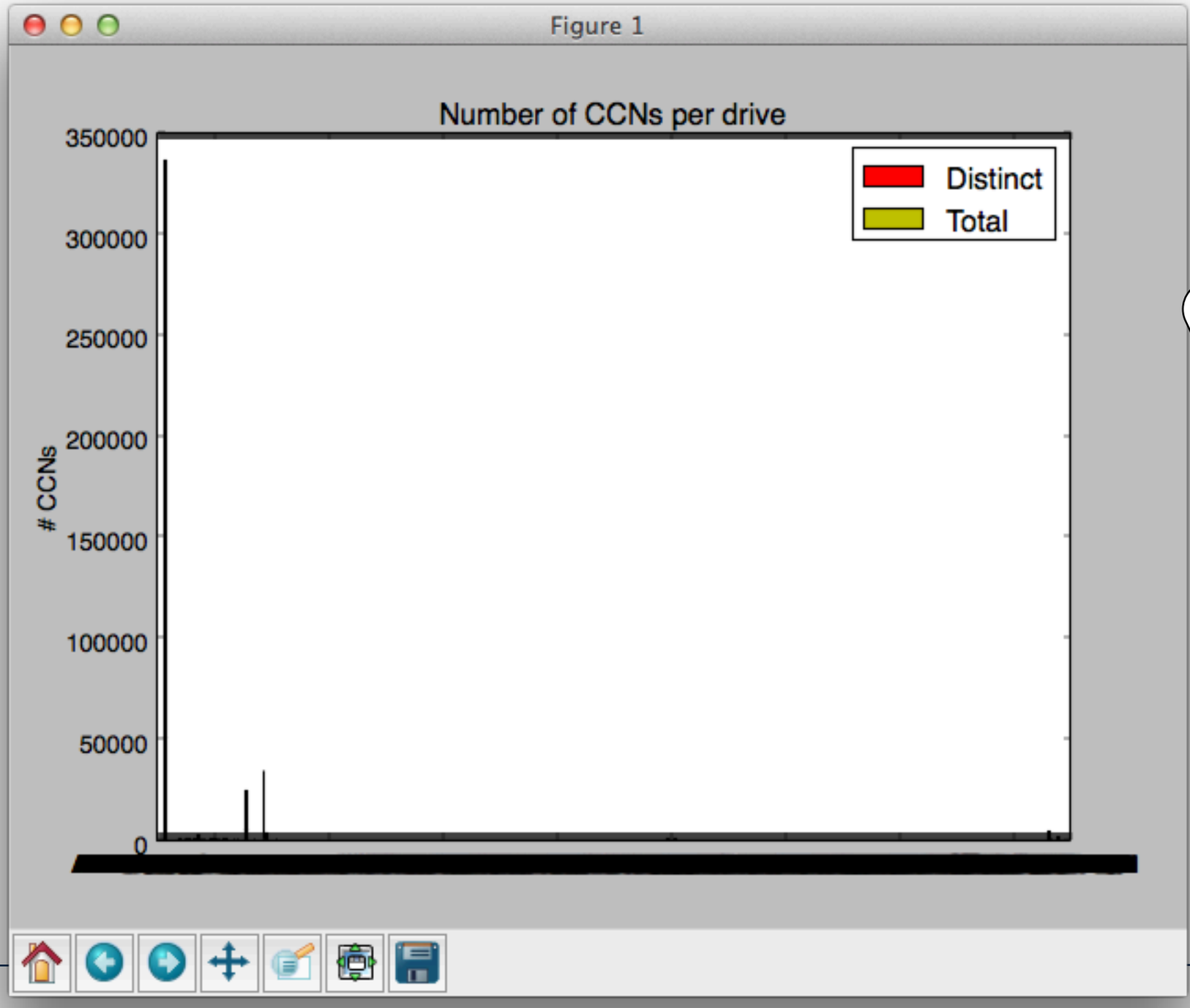
- Split axes — good for scale
  - —*c.f. logarithmic*
  - —*150 drives — easier to read*
- Data range 0-32,000
- 5 hours of work
- ≈50 lines of code

- 1000 drives
- Much larger data range (how much?)
- 20 min of work

Fix your graph one issue at a time.

Add the Y units by commenting out plt.yticks():

```
#plt.yticks(np.arange(0,81,10))
```



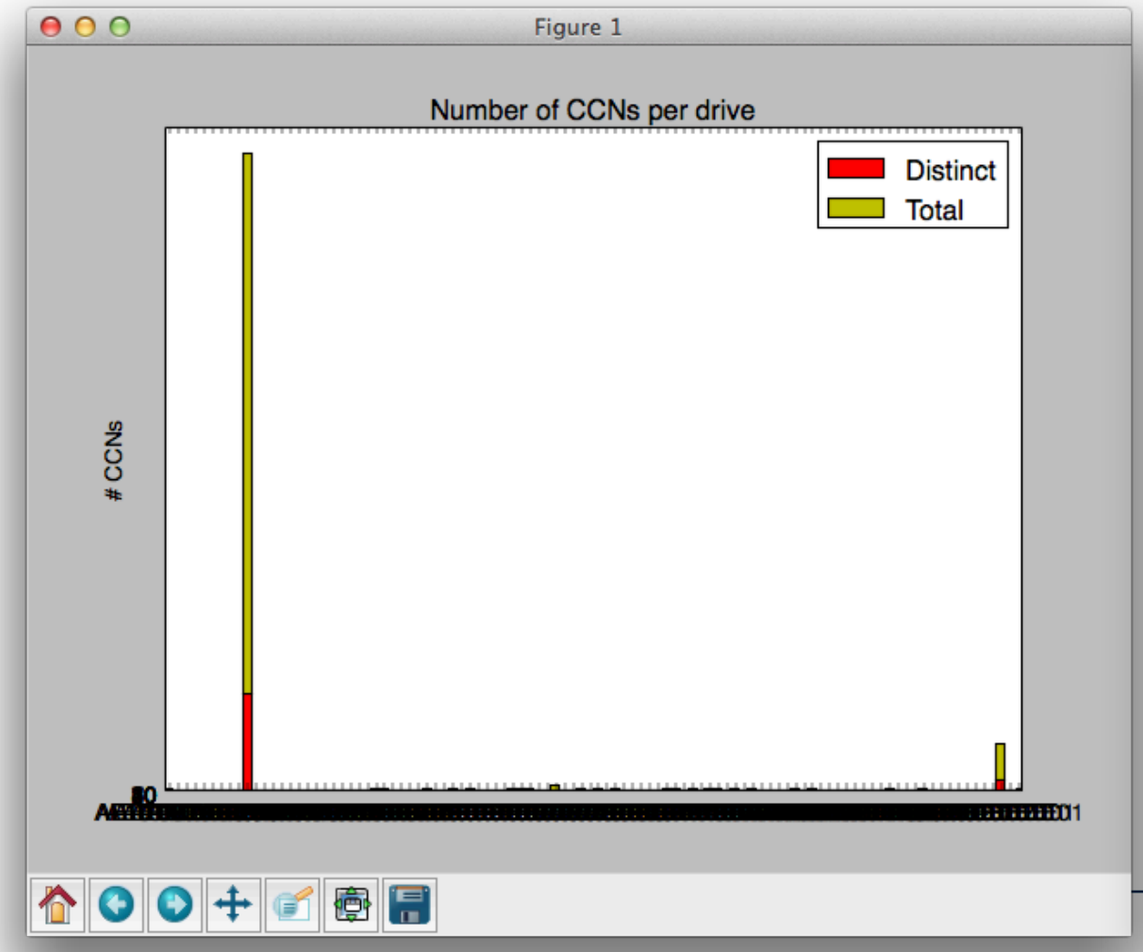
A semi-log plot does a better job showing the range.

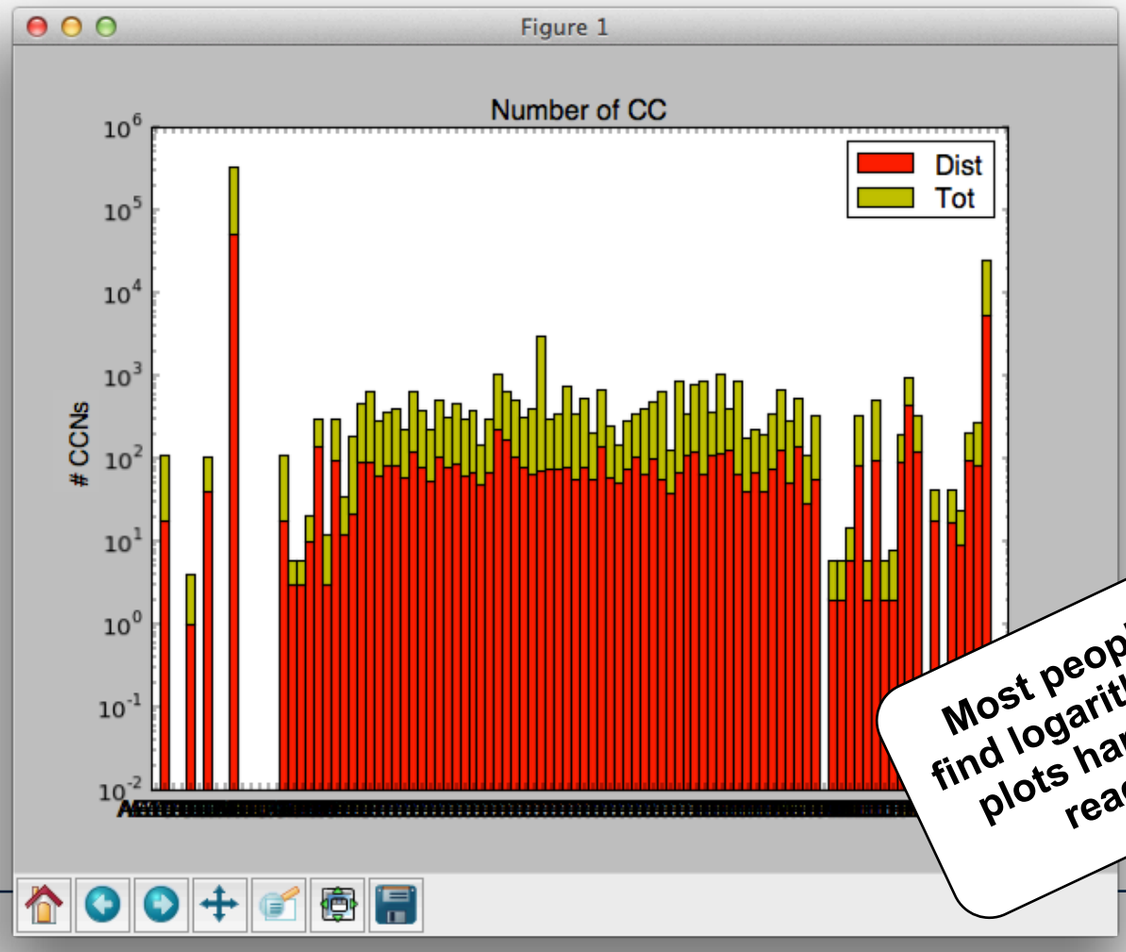
Change this:

```
p1 = plt.bar(ind, distinctCounts, width, color='r')
p2 = plt.bar(ind, totalCounts,    width, color='y',bottom=distinctCounts)
```

To this:

```
# for log plot, never go down to 0
bottom = [.01]*count
p1 = plt.bar(ind, distinctCounts, width, color='r',bottom=bottom)
p2 = plt.bar(ind, totalCounts,    width, color='y',bottom=distinctCounts)
plt.yscale('log')
```

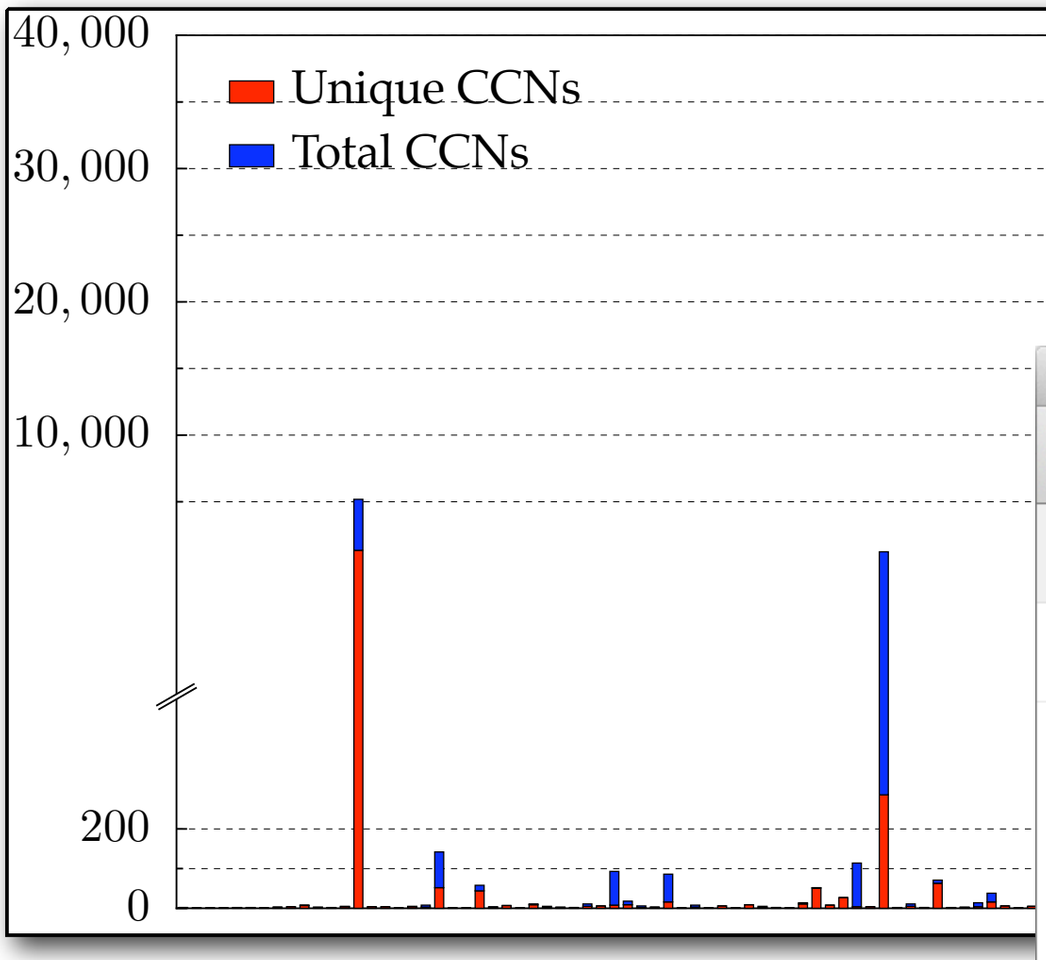


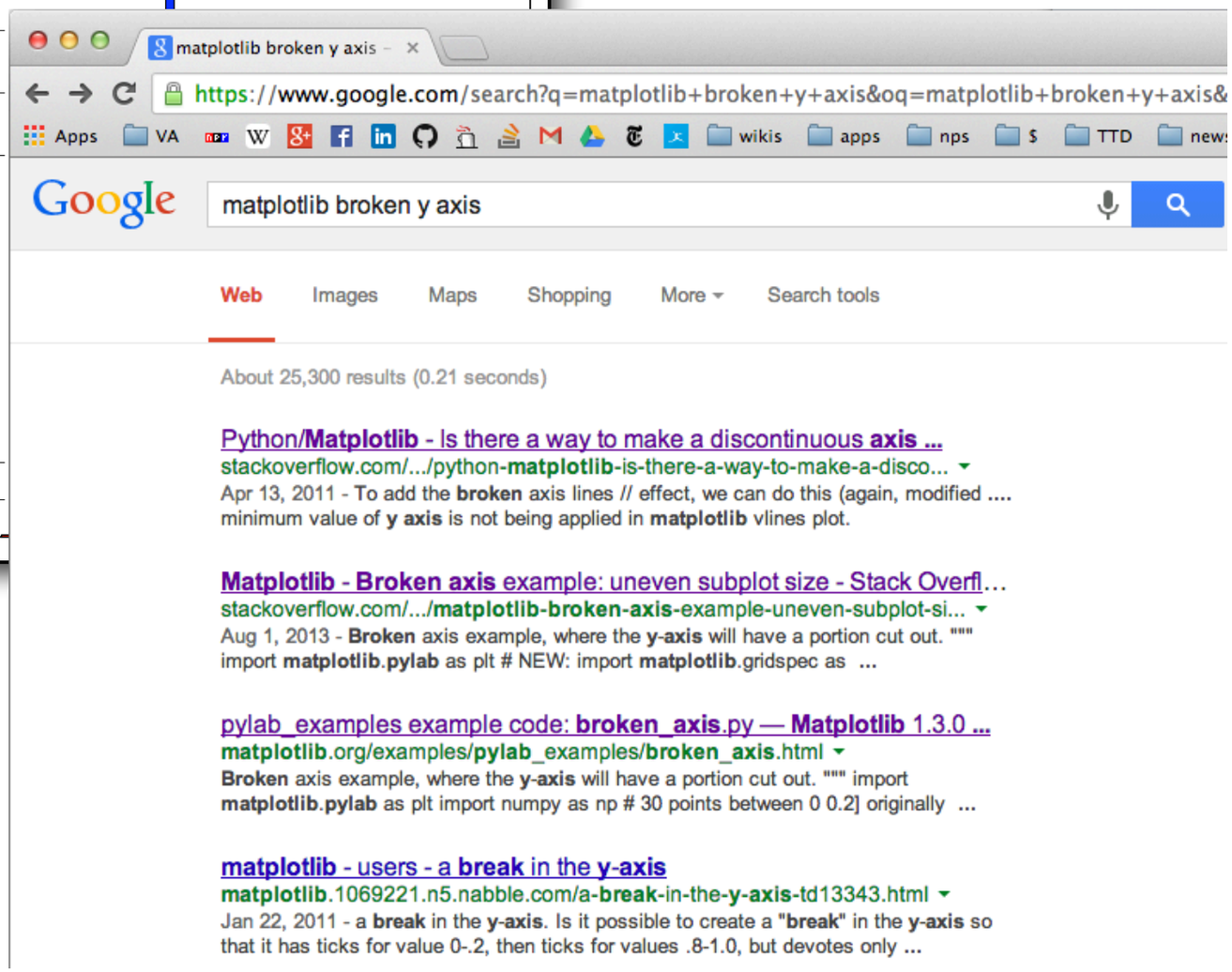


Most people find logarithmic plots hard to read.

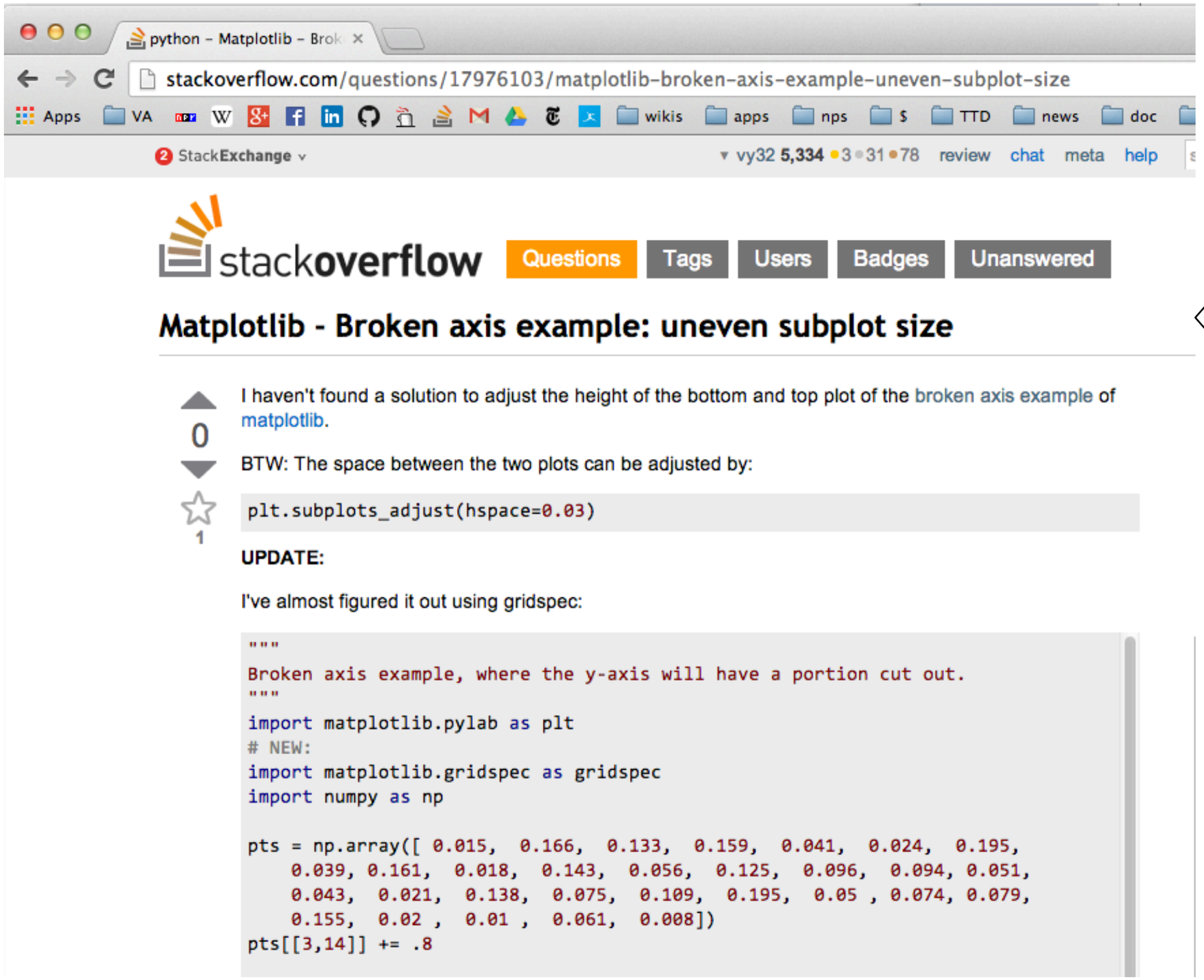
The original plot had a “broken axis.”

Use Google to find an example.

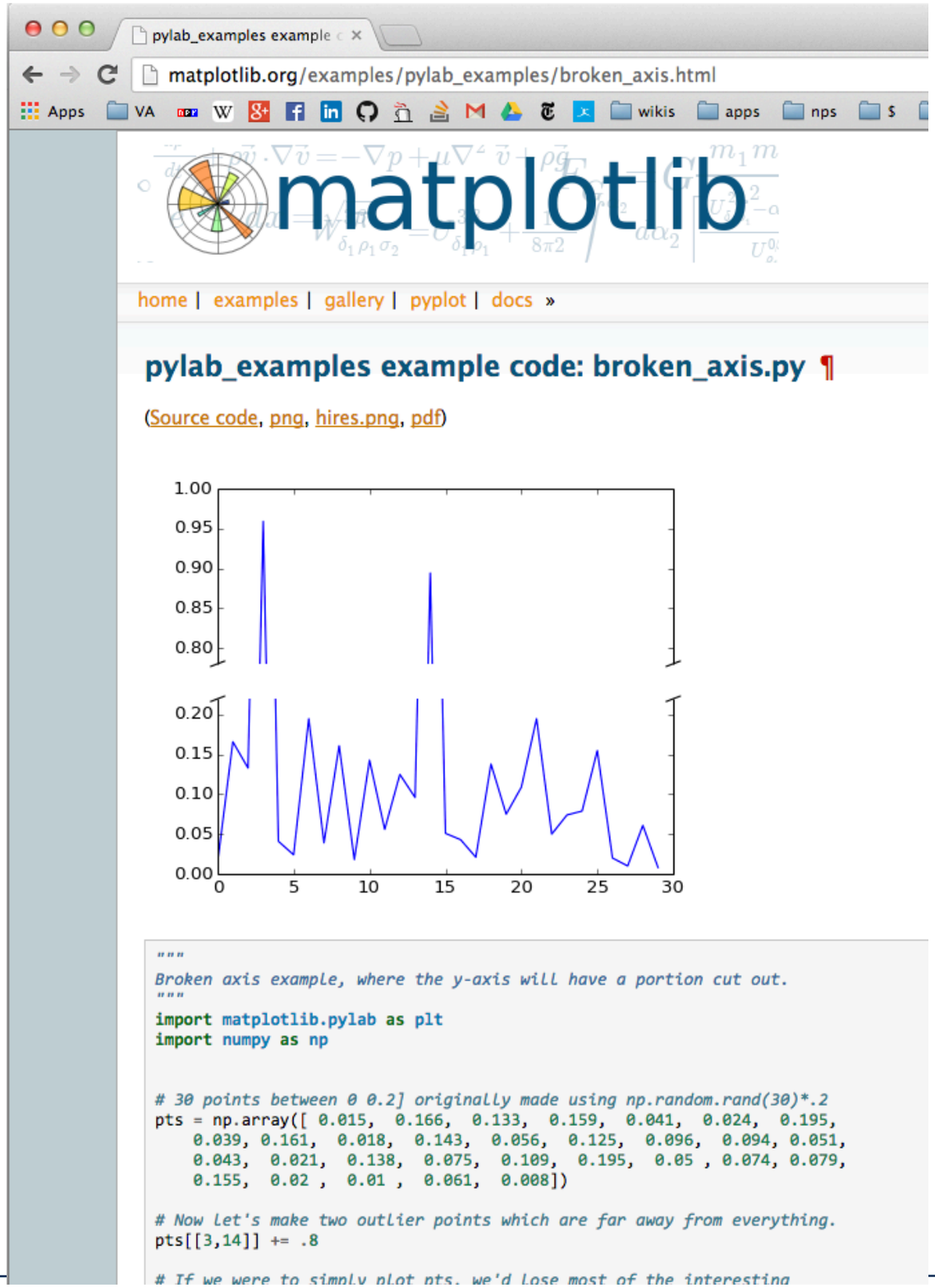




I quickly found an online example.



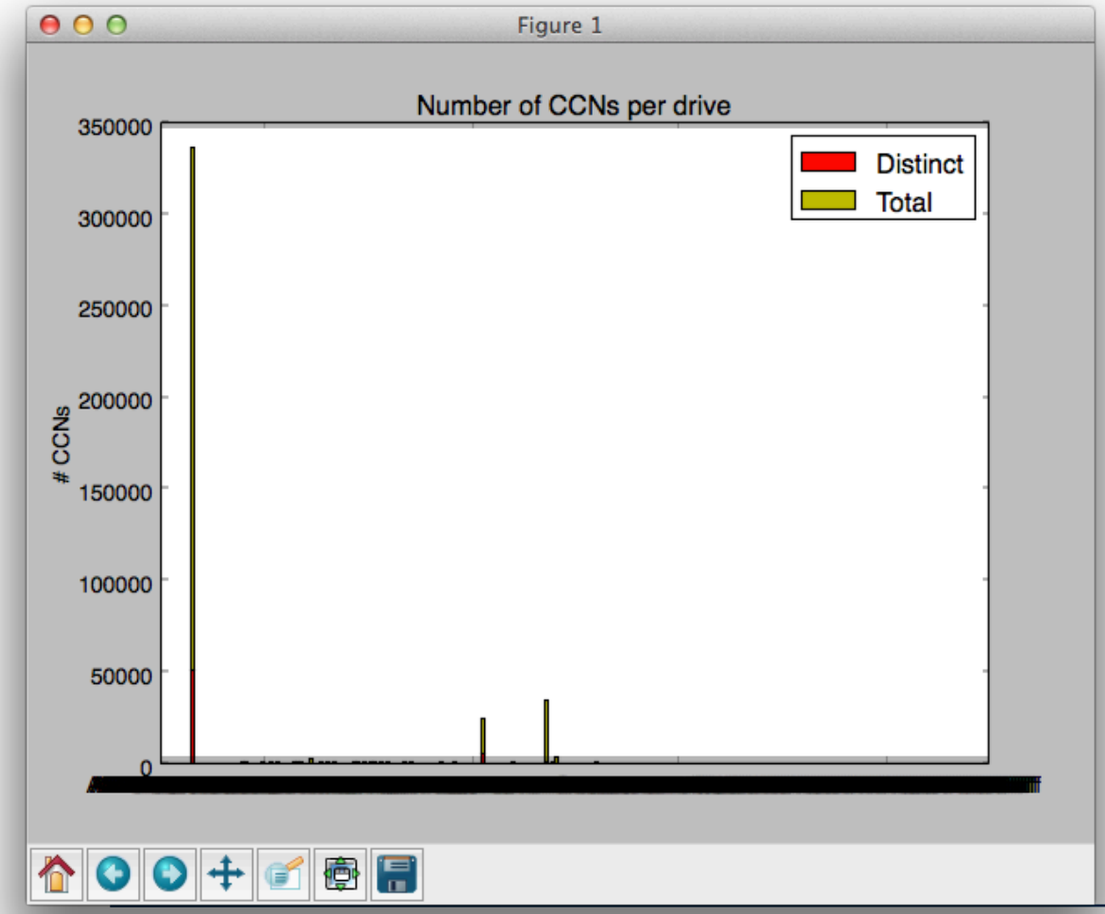
The broken axis is two plots with the same data and different zooms

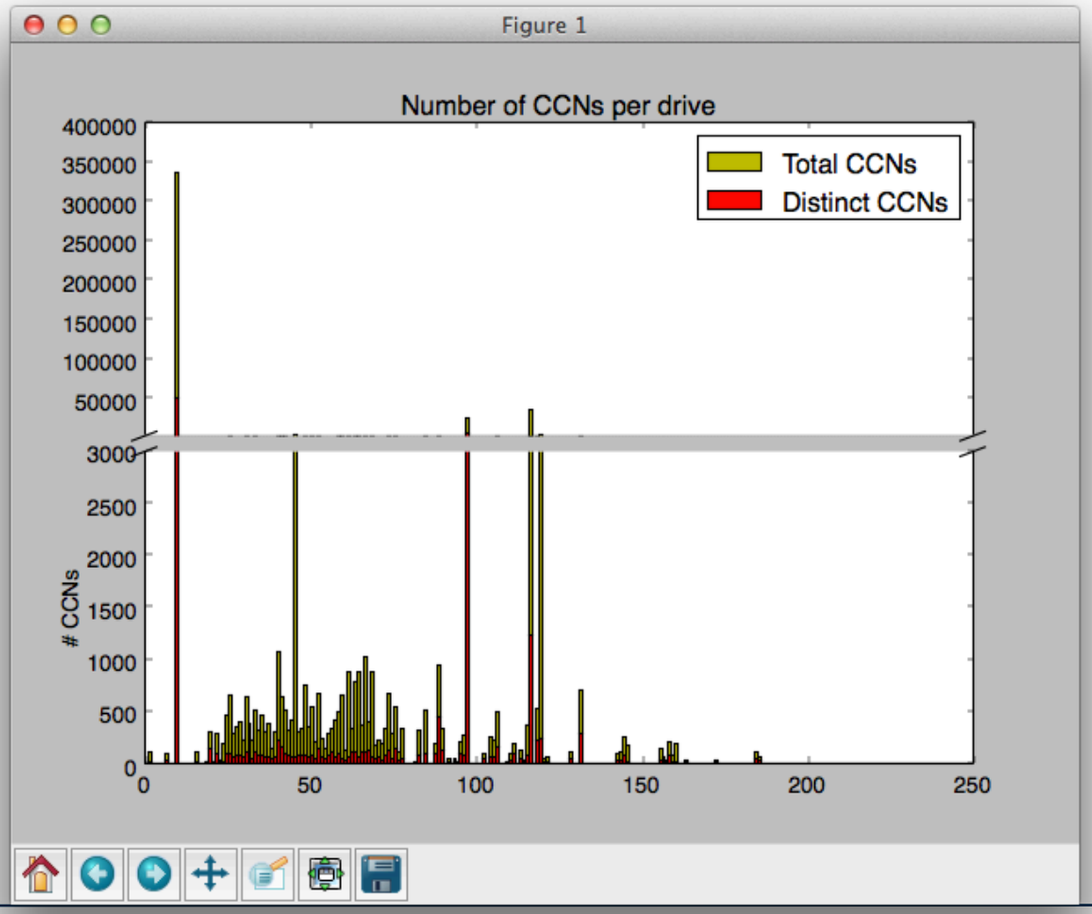



```
"""
Broken axis example, where the y-axis will have a portion cut out.
"""
import matplotlib.pylab as plt
import numpy as np


# 30 points between 0 0.2] originally made using np.random.rand(30)*.2
pts = np.array([ 0.015,  0.166,  0.133,  0.159,  0.041,  0.024,  0.195,
    0.039, 0.161,  0.018,  0.143,  0.056,  0.125,  0.096,  0.094, 0.051,
    0.043,  0.021,  0.138,  0.075,  0.109,  0.195,  0.05 , 0.074, 0.079,
    0.155,  0.02 ,  0.01 ,  0.061,  0.008])

# Now let's make two outlier points which are far away from everything.
pts[[3,14]] += .8

# If we were to simply plot pts, we'd lose most of the interesting
```

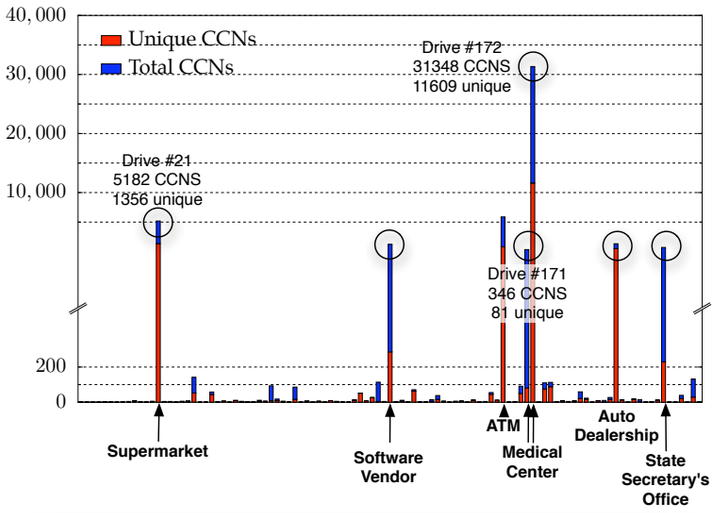
Create two plots:

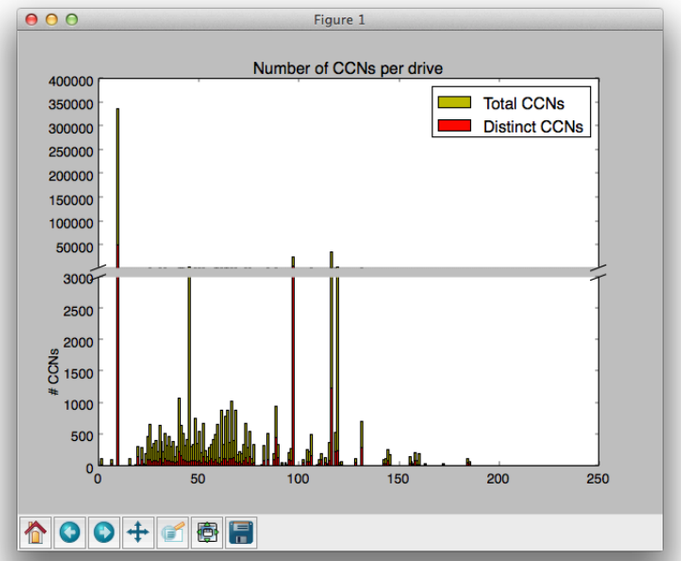
```
f,(ax,ax2) = plt.subplots(2,1,sharex=True)
..
ax.bar(ind, distinctCounts, width, color='r',bottom=bottom)
ax.bar(ind, totalCounts,    width, color='y',bottom=distinctCounts)
..

ax2.bar(ind, distinctCounts, width, color='r',bottom=bottom)
ax2.bar(ind, totalCounts,    width, color='y',bottom=distinctCounts)
...
```

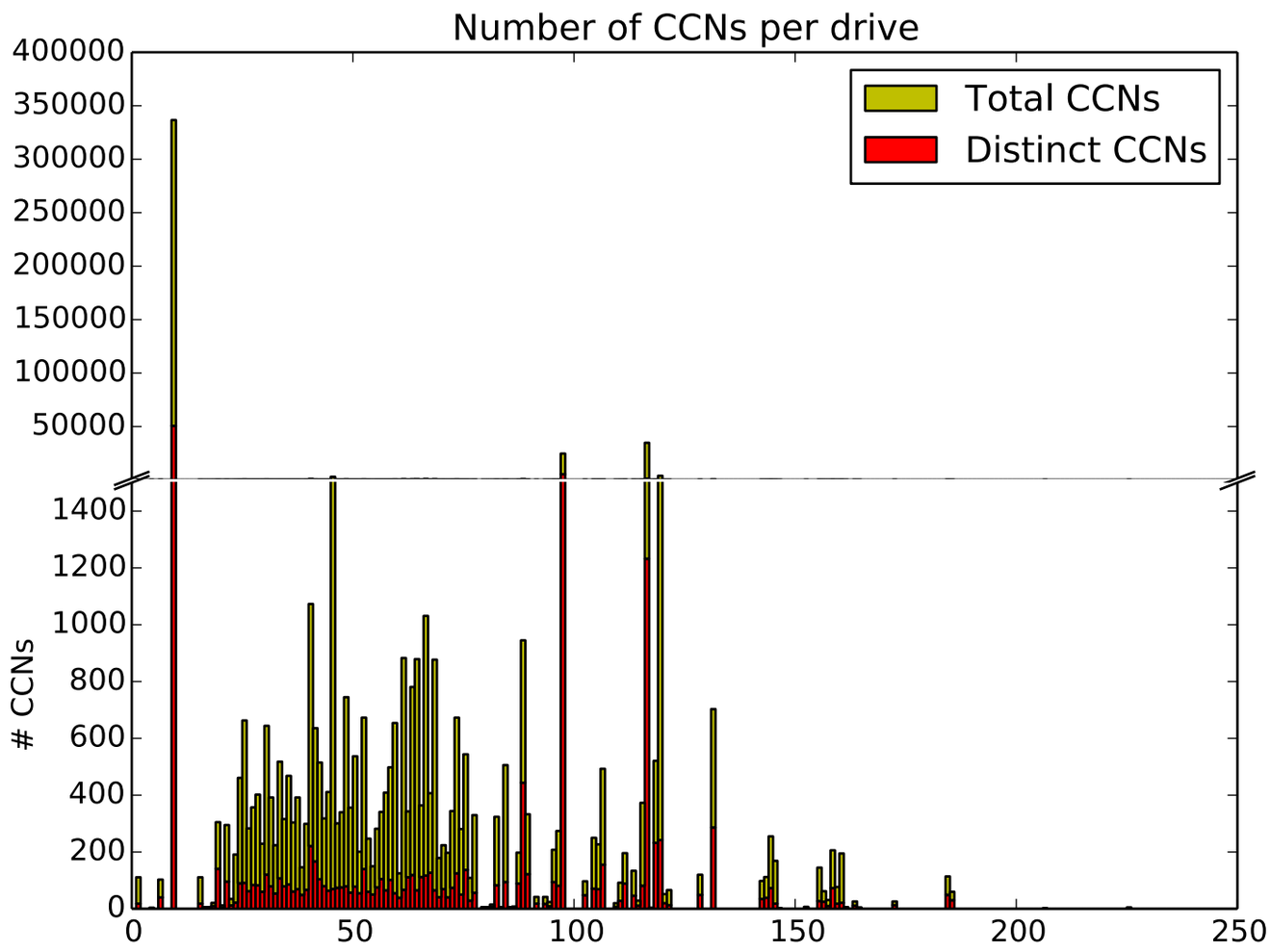




Change the output format to PDF.
(Bitmaps are a lousy way to show graphical information.)







Still need to do:

- Add commas to Y axis formatter
- Draw y grid lines across page
- Annotate exciting marks

PNG

WMF

PDF

JPEG

SVG

MP4

Output Formats

Visualization engines support multiple output formats

Bitmaps:

- GIF — Graphic Interchange Format
- PNG — Portable Network Graphics
- JPEG — Joint Photographic Experts Group
  - —*Don't output to bitmaps if you can help it*
  - —*Problems with zooming & blurring*

Line art:

- SVG — Scalable Vector Graphics
- PDF — Portable Document Format

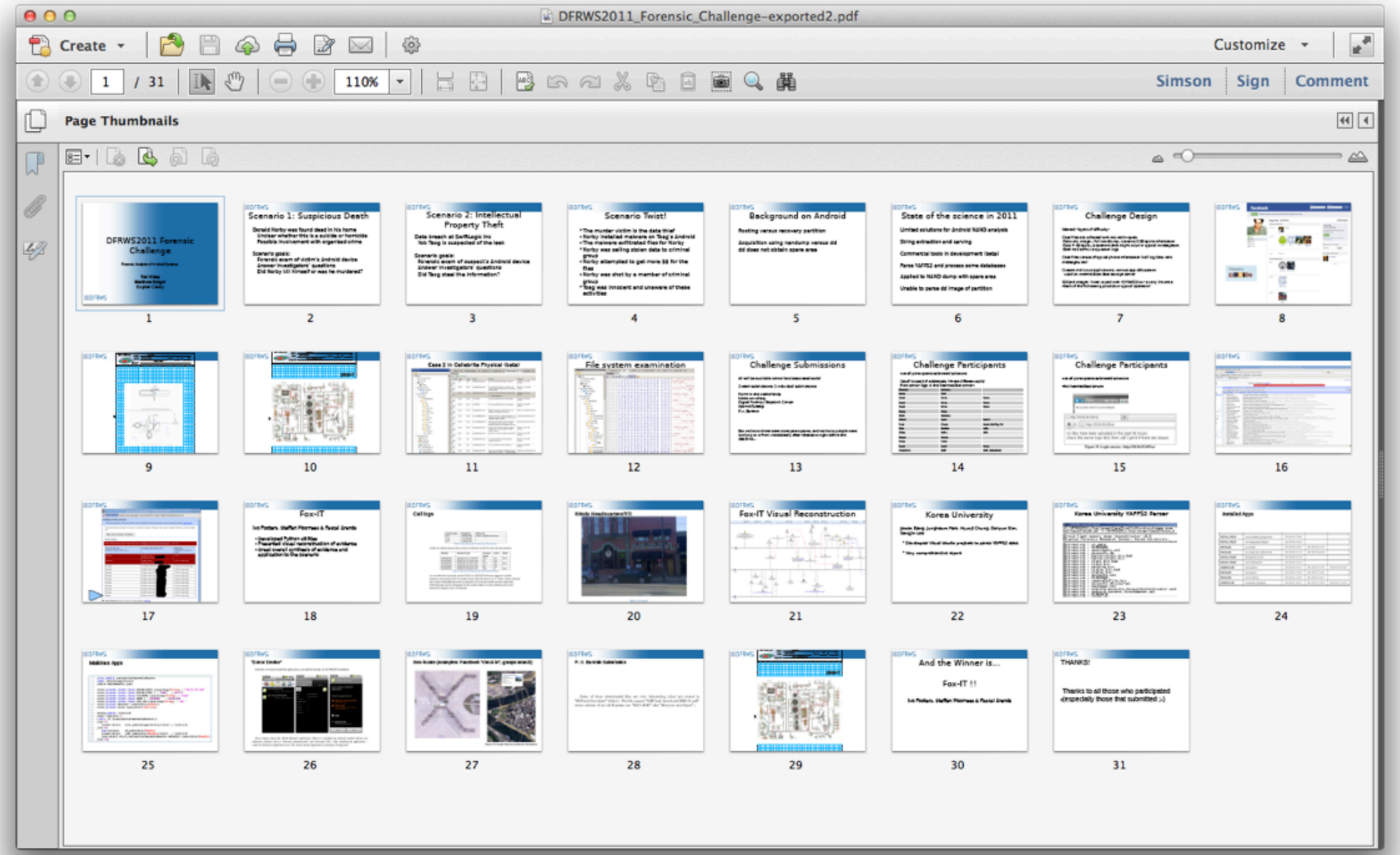
Animation:

- MOV — QuickTime
- SWF — Adobe Flash

PDF is a container format.
It can distribute a single image or multiple pages.

The FoxIT illustration was extracted from the DFRWS 2011 PDF:

`1690681 DFRWS2011_Forensic_Challenge-exported2.pdf`

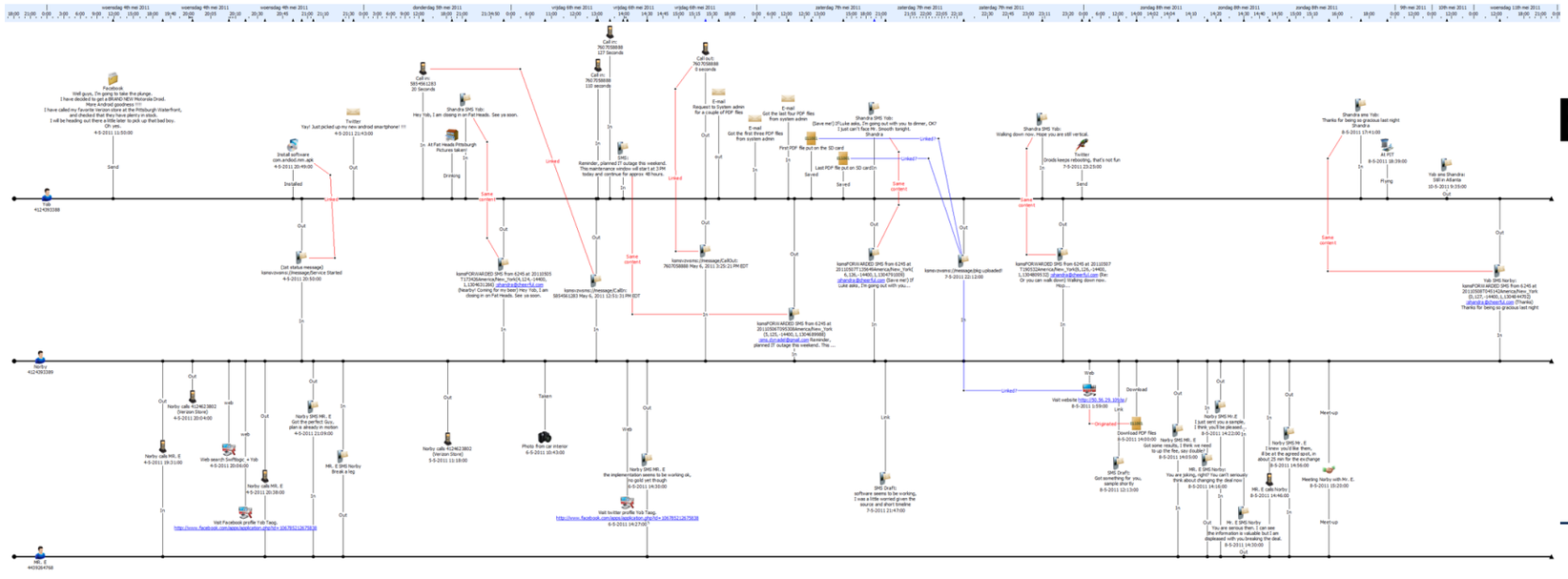


The relevant image is on p. 21

PDF content can be line art or bitmaps.

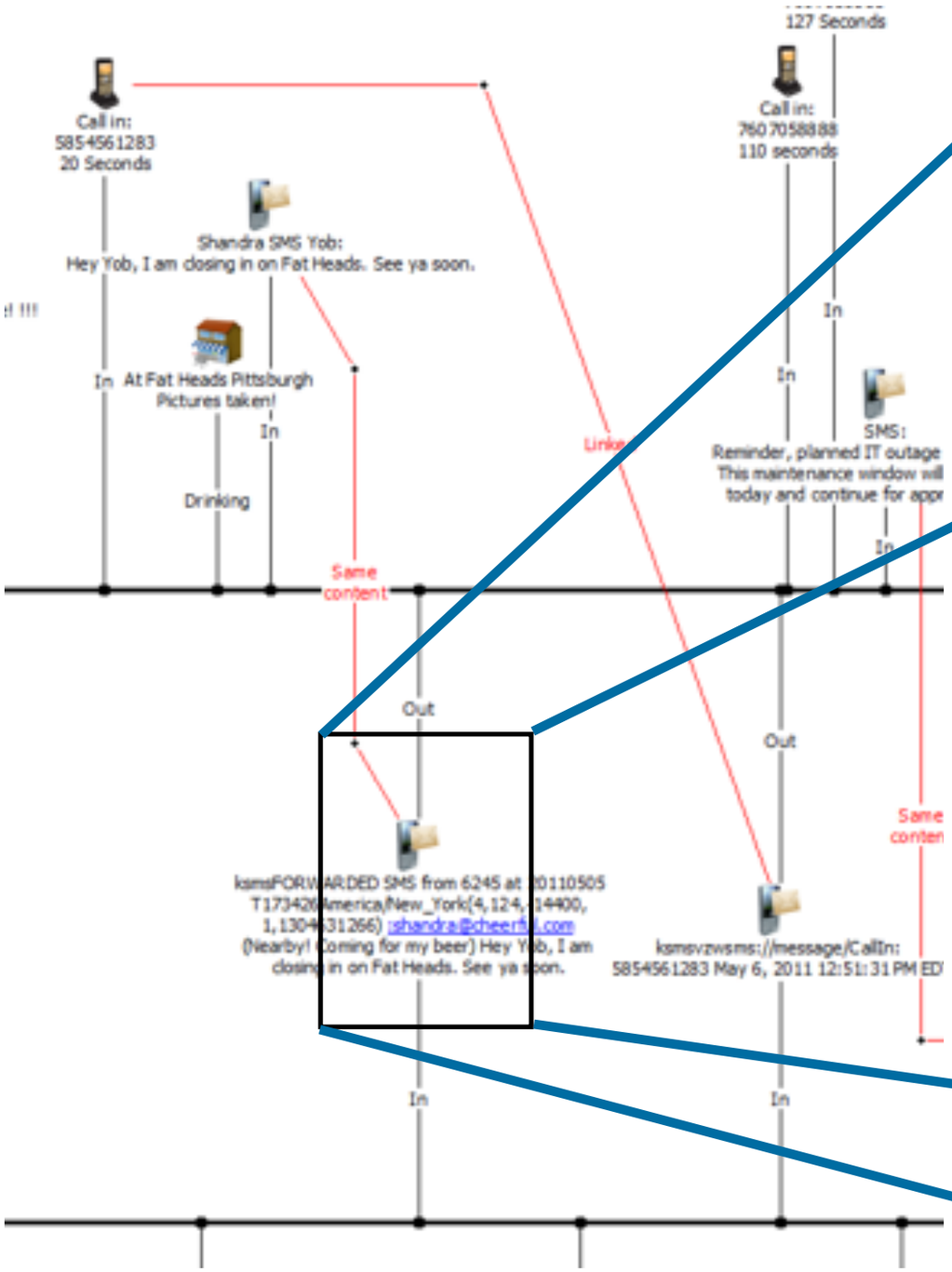
The FoxIT illustration is a bitmap. Extract it with ***pdfimages***:

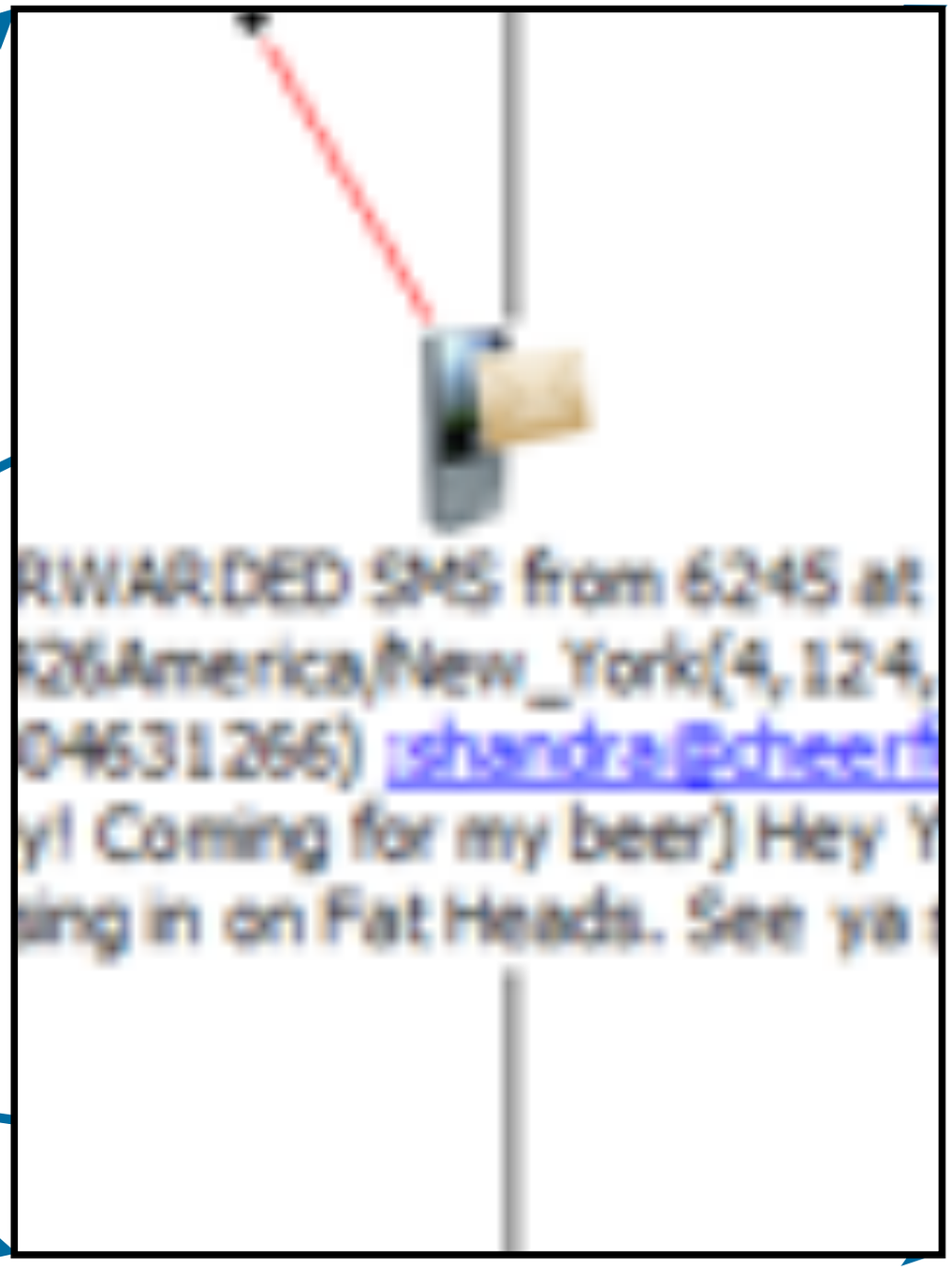
```
$ pdfimages DFRWS2011_Forensic_Challenge-exported2.pdf \
  -f 21 -l 21 -j foxit
$ BLOCKSIZE=1024 ls -s1
total 16832
 1652 DFRWS2011_Forensic_Challenge-exported2.pdf
    4 foxit-000.jpg
   40 foxit-001.ppm
    4 foxit-002.pbm
15132 foxit-003.ppm
$ convert foxit-003.ppm foxit-003.png
$ BLOCKSIZE=1024 ls -s1 foxit-003.png
264 foxit-003.png
$
```



267 KB bitmap
Extracted with pdfimages
converted with
ImageMagick

Zoomed in, we still have a bitmap.
Bitmaps are not “accessible” and can’t be searched.

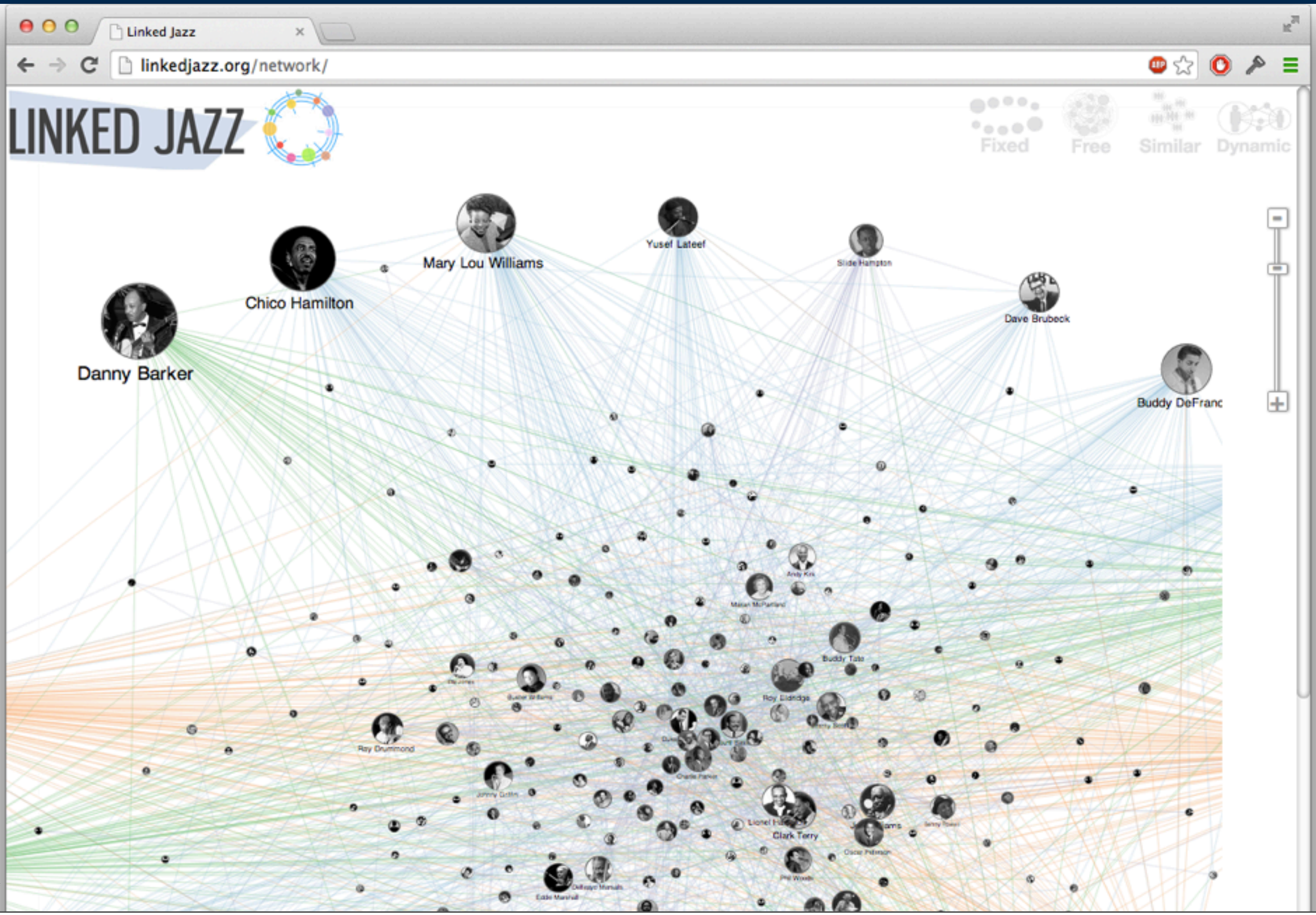




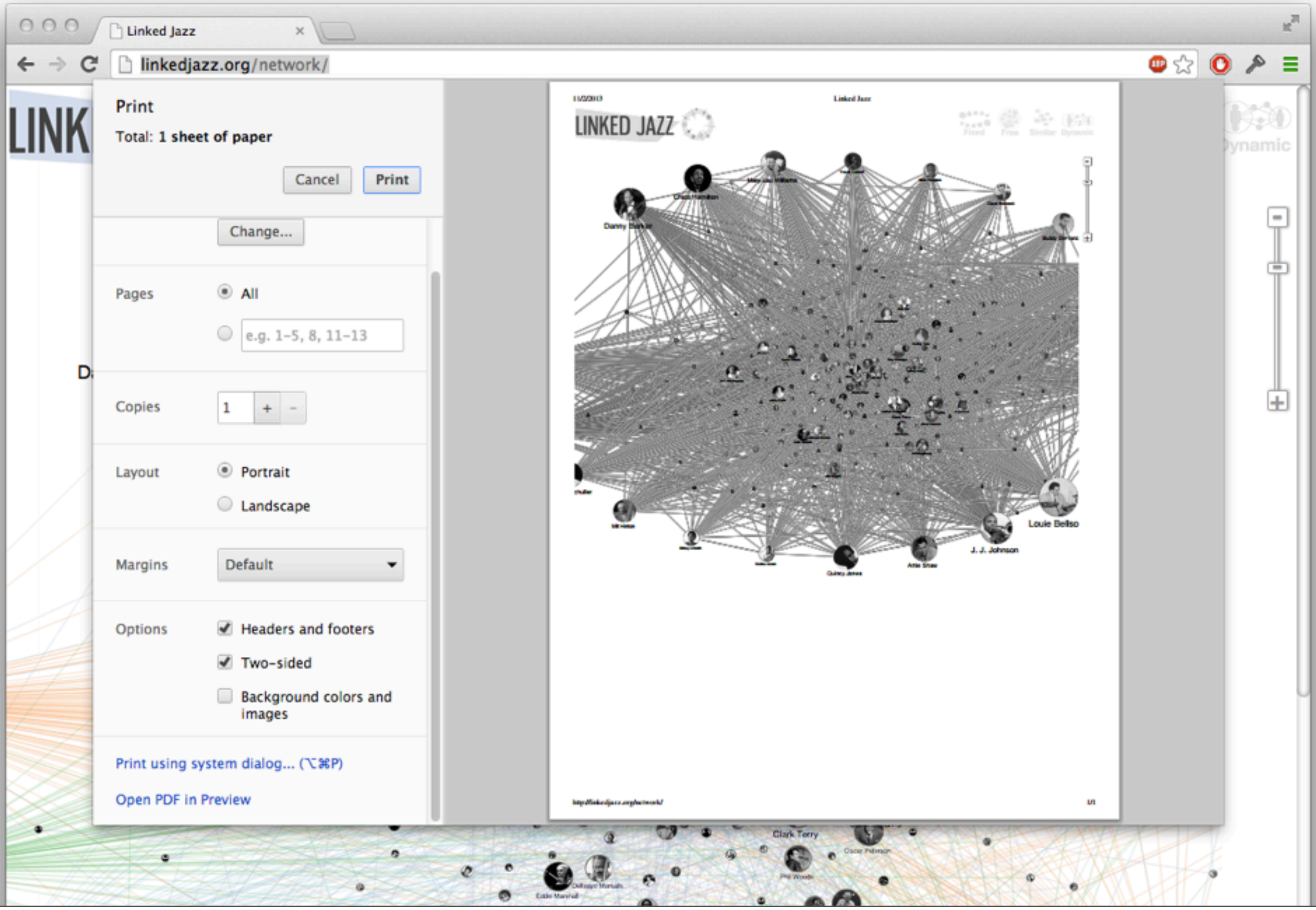
4x magnification

16x magnification

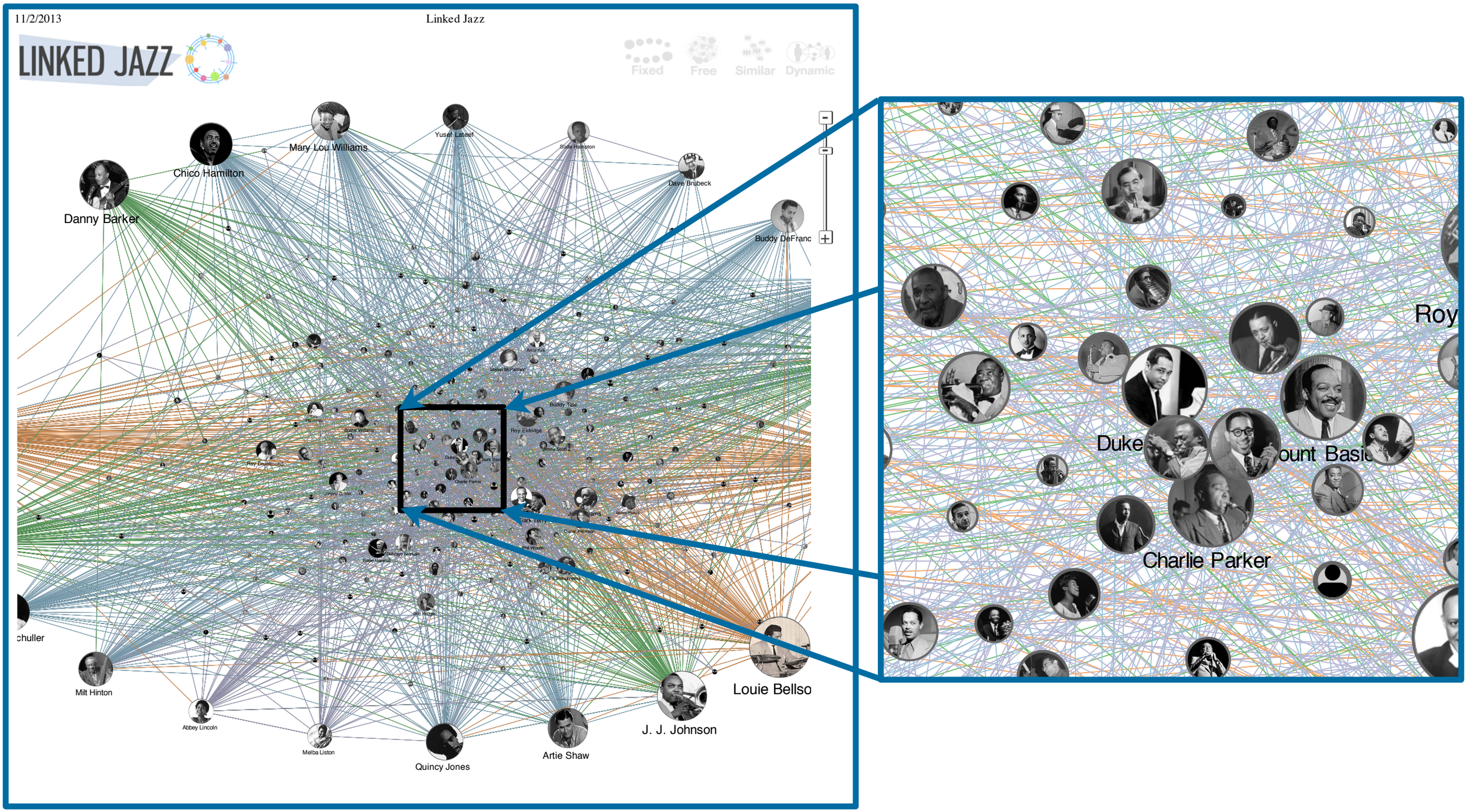
Many of the JavaScript libraries produce SVG output



SVG can be transformed to PDF with the browser's "print" command.

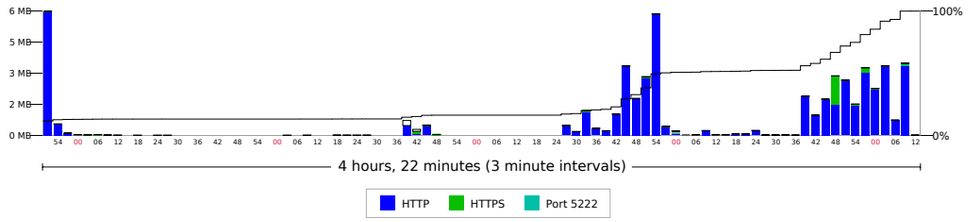


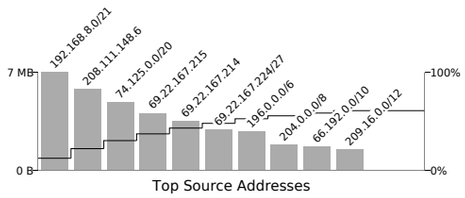
The resulting PDF is 2.6MB.
You can zoom and search for text.



TCPFLOW 1.4.0
Input: /corp/nps/packets/2008-nitroba/nitroba.pcap
Generated: 2013-10-06 21:20:44

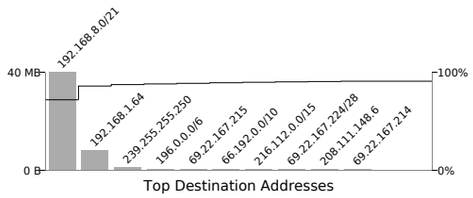
Date range: 2008-07-21 21:51:07 -- 2008-07-22 02:13:47
Packets analyzed: 91,144 (55.02 MB)
Transports: IPv4 100%





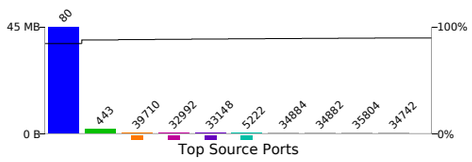
Top Source Addresses

1) 192.168.8.0/21 - 6.56 MB (11%)
2) 208.111.148.6 - 5.41 MB (9%)
3) 74.125.0.0/20 - 4.55 MB (8%)



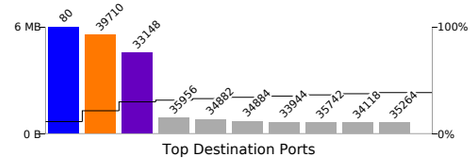
Top Destination Addresses

1) 192.168.8.0/21 - 39.58 MB (71%)
2) 192.168.1.64 - 7.77 MB (14%)
3) 239.255.255.250 - 818.61 KB (1%)



Top Source Ports

1) 80 - 44.78 MB (84%)
2) 443 - 1.82 MB (3%)
3) 39710 - 178.85 KB (0%)



Top Destination Ports

1) 80 - 5.85 MB (11%)
2) 39710 - 5.41 MB (10%)
3) 33148 - 4.44 MB (8%)

netviz for tcpflow

netviz is a network visualization that we added to tcpflow

Design goals:

- Handle any number of packets
- Output in PDF
- Easy to use without training
- Use the BE13 API

Input: 1 or more PCAP files

- 1–1G packets
- 1–1G connections
- 1–4Gi hosts

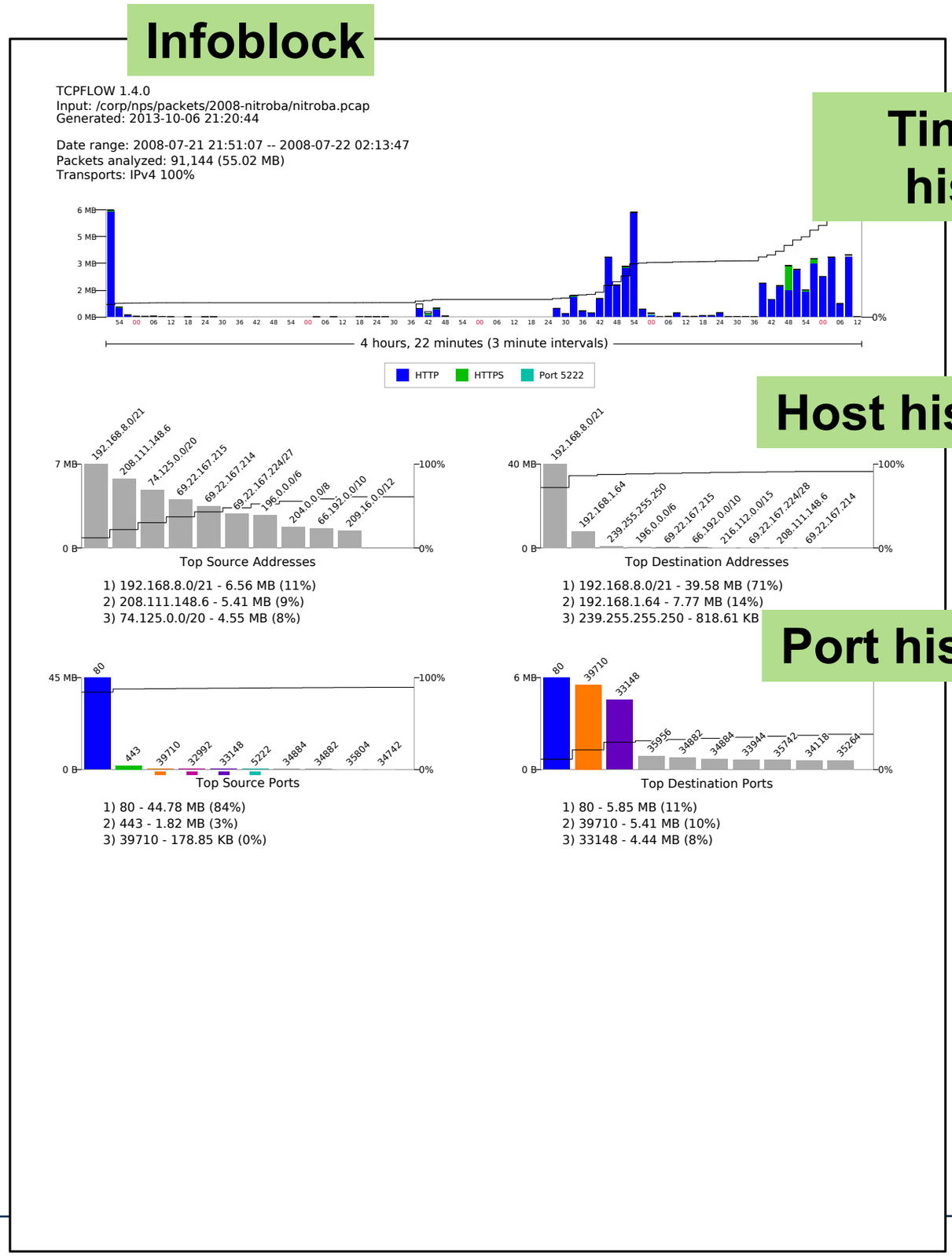
Output: PDF file of ≈ 50KB

Infoblock

Time-based histogram

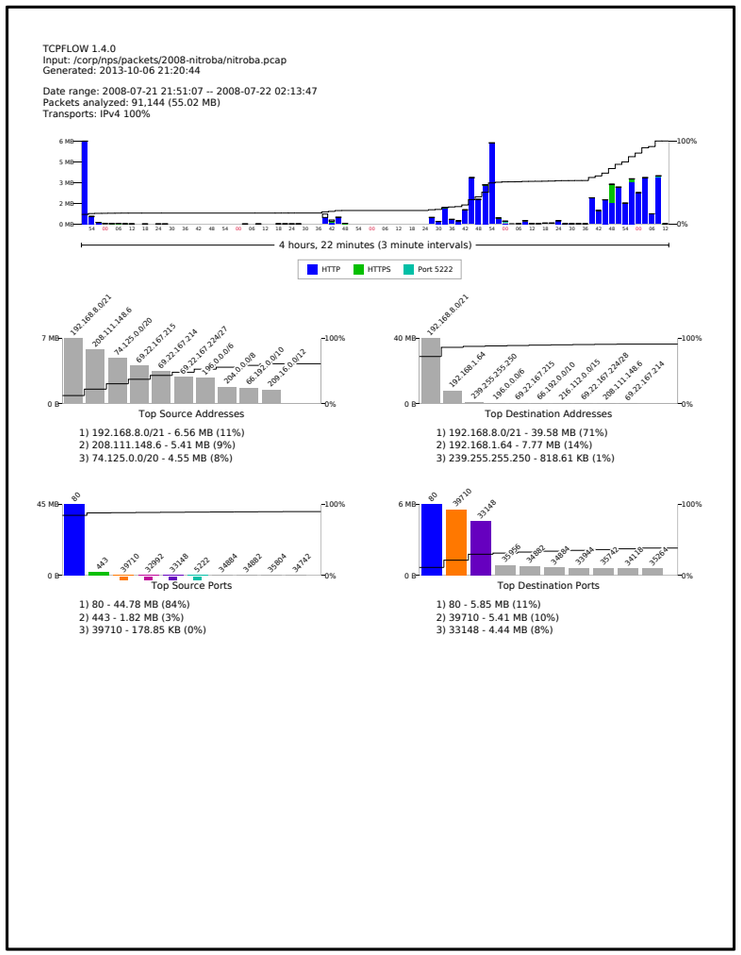
Host histograms

Port histograms



Netviz is implemented with open source libraries.

Output Format	PDF
Output Engine	Cairo
Layout and Typography	Custom (mistake?)
Implementation	C++



Other options we considered:

- LaTeX
- HTML & SVG
- HTML5 (JavaScript & Canvas)

We were trying to minimize dependencies

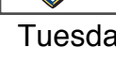
Infoblock provides important forensic information.

Always label your visualizations with:

- Input
- Date of input file
- Date visualization was run
- Command line used to generate the output (we forgot this)

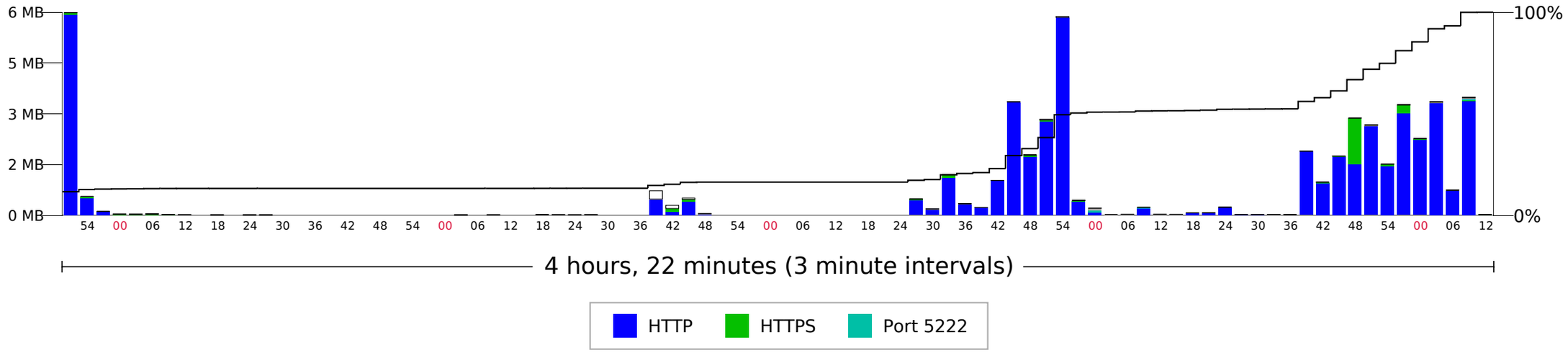
```
TCPFLOW 1.4.0
Input: /corp/nps/packets/2008-nitroba/nitroba.pcap
Generated: 2013-10-06 21:20:44

Date range: 2008-07-21 21:51:07 -- 2008-07-22 02:13:47
Packets analyzed: 91,144 (55.02 MB)
Transports: IPv4 100%
```

Labeling is important for repeatability & admissibility

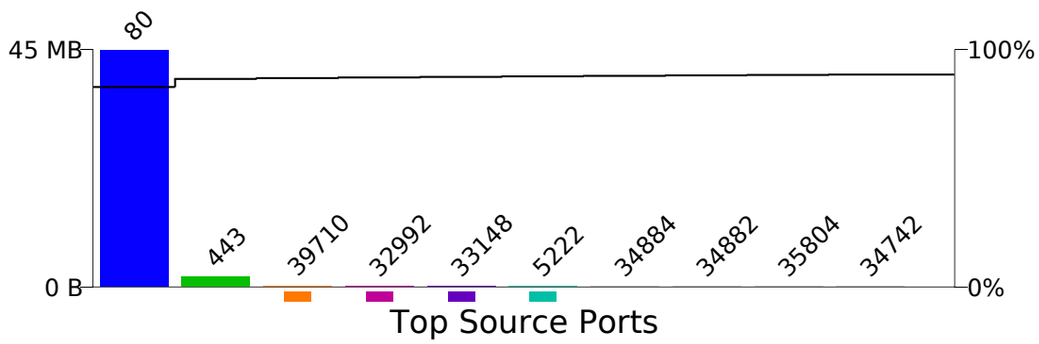


The time-based histogram shows how many packets were received, and when they were received.

The histogram lets an analyst make a rapid determination about what's in a PCAP file.

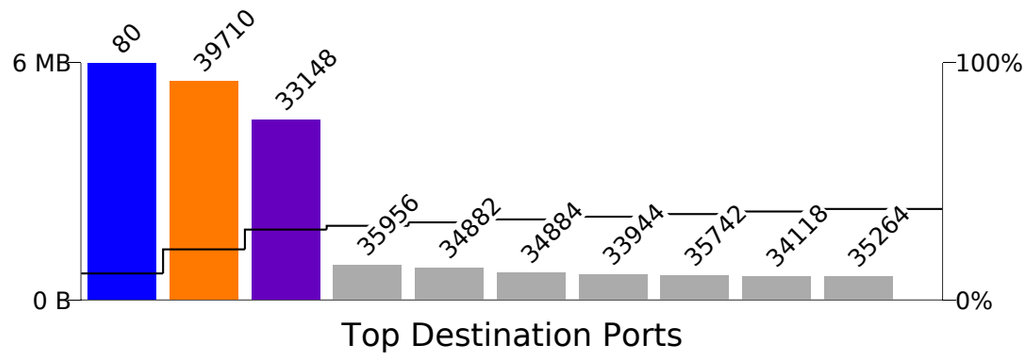


Port histograms that show sources & destinations. These use the same code as the time-based histogram.



Top Source Ports

1) 80 - 44.78 MB (84%)
2) 443 - 1.82 MB (3%)
3) 39710 - 178.85 KB (0%)



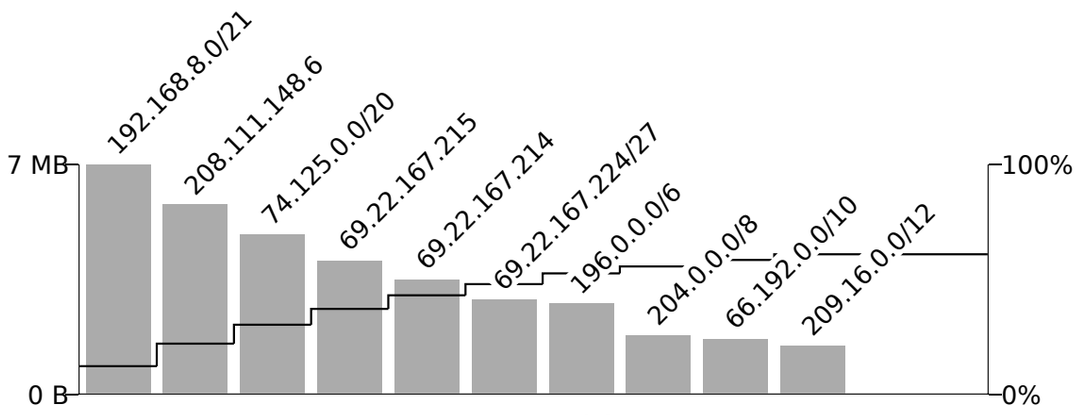
Top Destination Ports

1) 80 - 5.85 MB (11%)
2) 39710 - 5.41 MB (10%)
3) 33148 - 4.44 MB (8%)

Note:

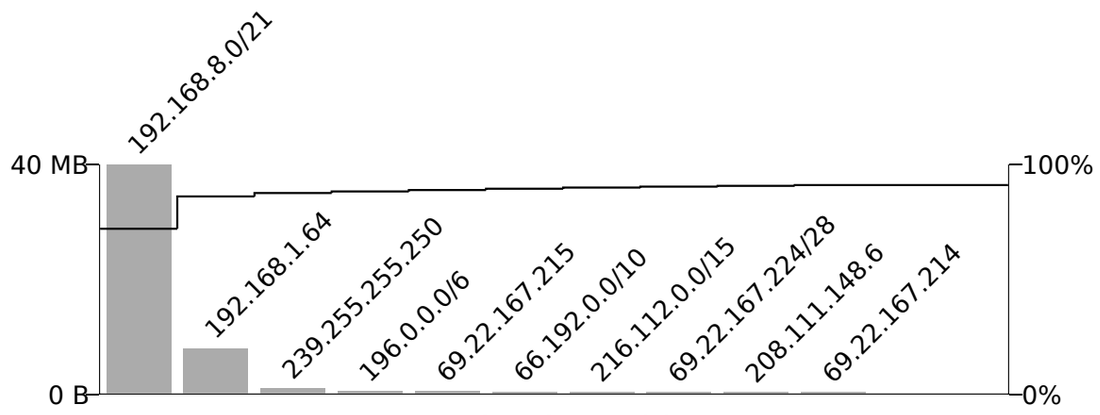
- Color key for the time histogram is presented on the port histogram.

Address histogram shows source & destination addresses.



Top Source Addresses

1) 192.168.8.0/21 - 6.56 MB (11%)
2) 208.111.148.6 - 5.41 MB (9%)
3) 74.125.0.0/20 - 4.55 MB (8%)



Top Destination Addresses

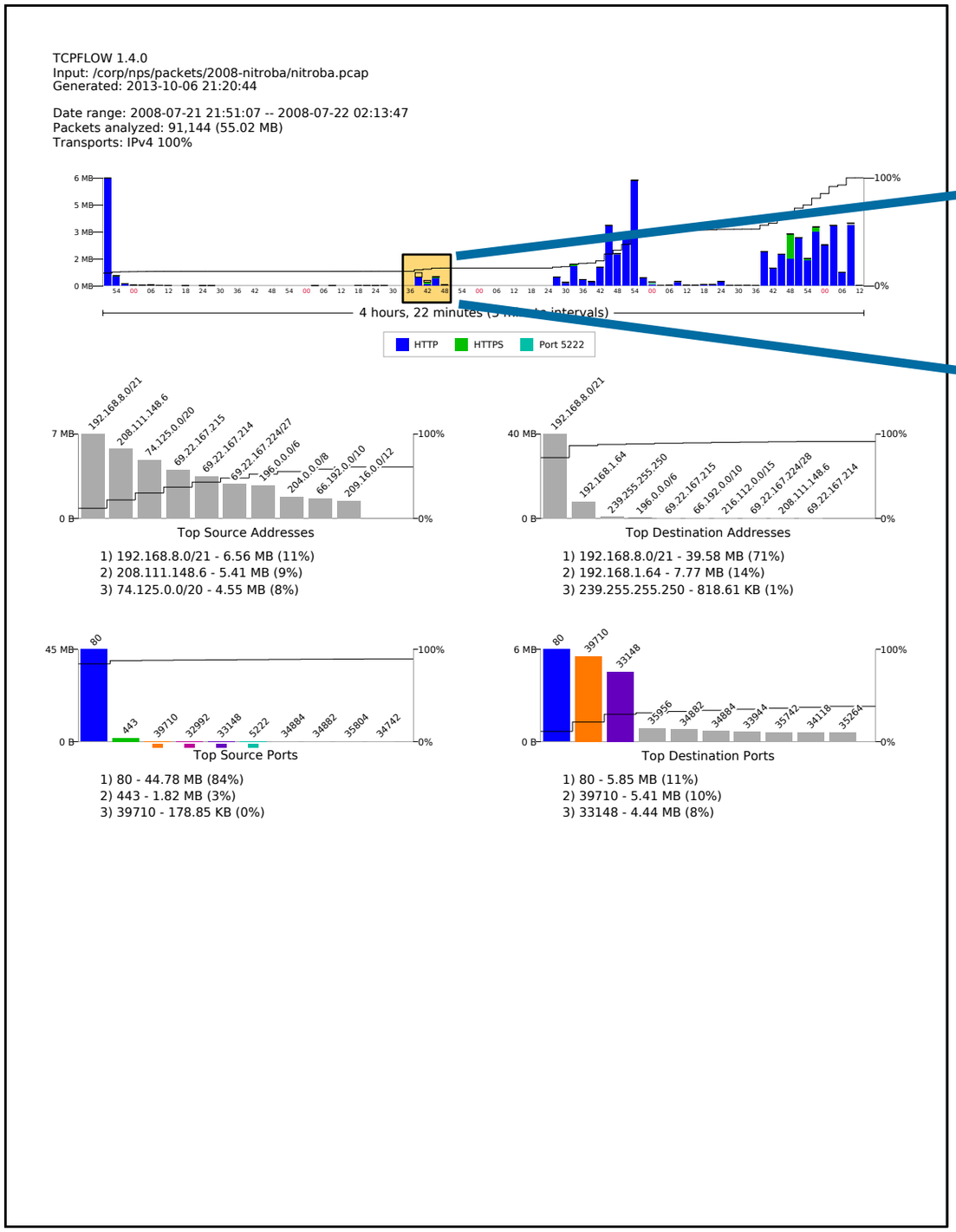
1) 192.168.8.0/21 - 39.58 MB (71%)
2) 192.168.1.64 - 7.77 MB (14%)
3) 239.255.255.250 - 818.61 KB (1%)

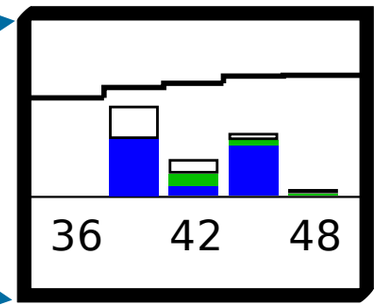
Problem: There might be 2^{32} IPv4 addresses or 2^{128} IPv6 addresses.

Solution: Tree-based counter

- Note 192.168.8.0/21 in above display — the /21 was automatically determined
- The tree has some performance problems that need to be addressed

PDF typically contains vector graphics, allowing for high resolution.





PDF supports 32 bit floats & ints

$8.5” \div 2^{32} \approx 1nm$ feature size

Other vector graphic formats:

- PostScript (PDF is based on PS)
- Windows Meta File (WMF)
- Scalable Vector Graphics (SVG)

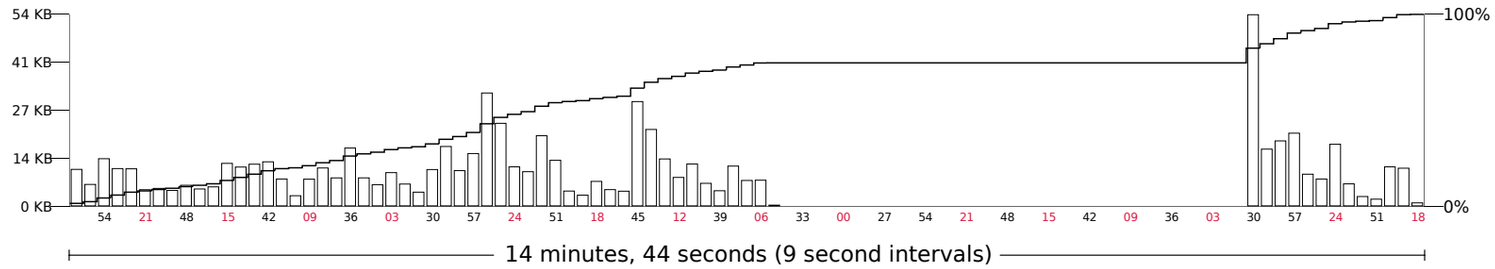
The goal of netviz is to give rapid “situational awareness” about a set of of packets.

This flow has:

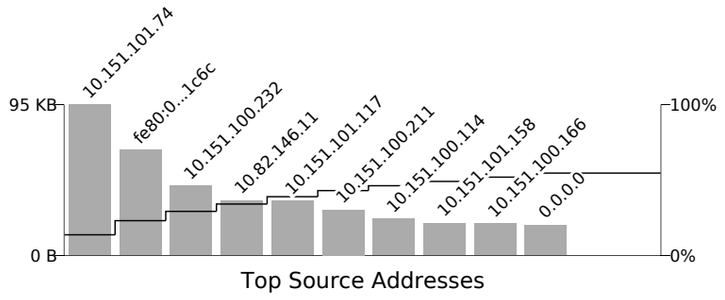
- No TCP
- 9% IPv6
- A big gap with no data

TCPFLOW 1.4.0
Input: out.pcap
Generated: 2013-09-18 11:48:20

Date range: 2013-09-18 11:29:42 -- 2013-09-18 11:44:25
Packets analyzed: 3,177 (704.80 KB)
Transports: IPv4 90% IPv6 9%

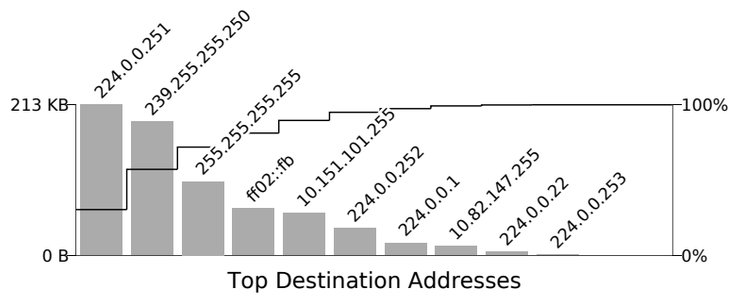


No TCP



Top Source Addresses

1) 10.151.101.74 - 95.02 KB (13%)
2) fe80::1240:f3ff:fe92:1c6c - 66.18 KB (9%)
3) 10.151.100.232 - 43.59 KB (6%)



Top Destination Addresses

1) 224.0.0.251 - 213.06 KB (30%)
2) 239.255.255.250 - 188.49 KB (26%)
3) 255.255.255.255 - 104.23 KB (14%)

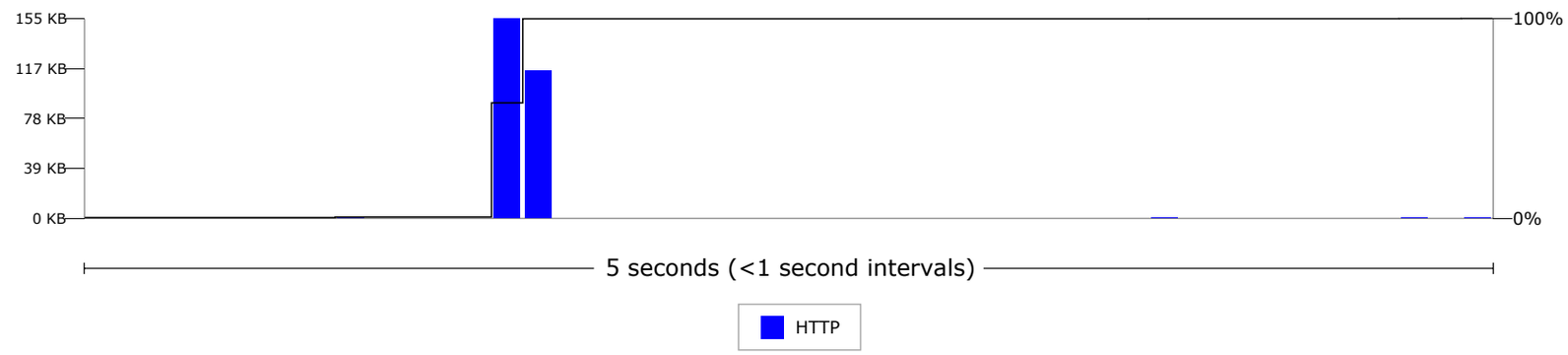
No Source Ports

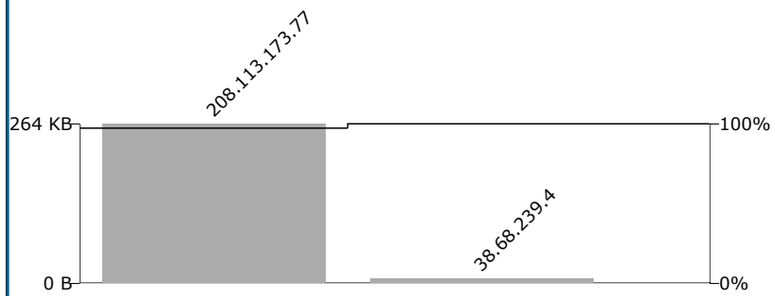
No Destination Ports

This flow has a single download of about a megabyte. It also has two failed HTTP requests.

TCPFLOW 1.4.0
Input: words.pcap
Generated: 2013-10-29 16:23:57

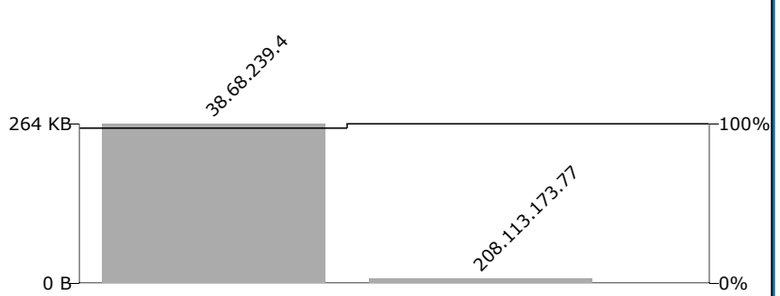
Date range: 2013-10-29 16:20:32 -- 2013-10-29 16:20:36
Packets analyzed: 319 (271.83 KB)
Transports: IPv4 100%





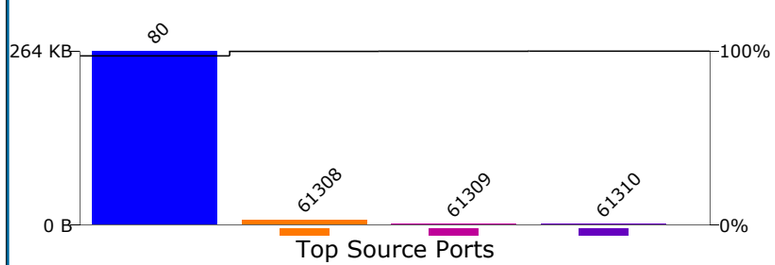
Top Source Addresses

1) 208.113.173.77 - 264.42 KB (97%)
2) 38.68.239.4 - 7.42 KB (2%)



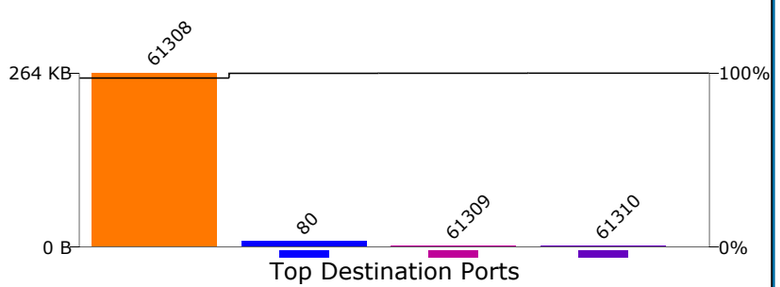
Top Destination Addresses

1) 38.68.239.4 - 264.42 KB (97%)
2) 208.113.173.77 - 7.42 KB (2%)



Top Source Ports

1) 80 - 264.42 KB (97%)
2) 61308 - 7.05 KB (2%)
3) 61309 - 186 B (0%)



Top Destination Ports

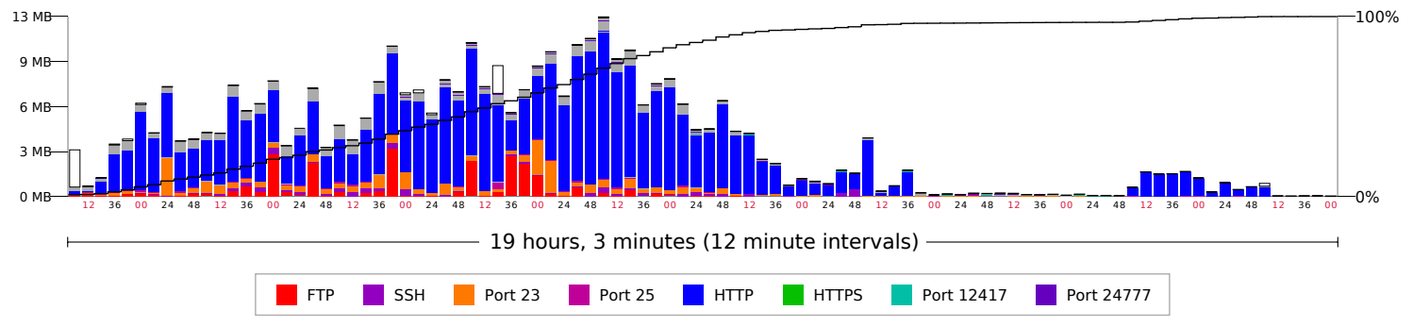
1) 61308 - 264.15 KB (97%)
2) 80 - 7.42 KB (2%)
3) 61309 - 132 B (0%)

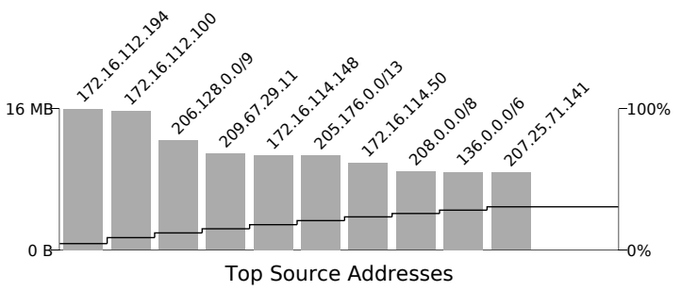
MIT ID'99 IDS evaluation.
The graph shows peculiarities of the traffic.

- Lots of FTP
- Not enough off-peak data.
- Lots of non-TCP

TCPFLOW 1.4.0
Input: /corp/mitll/packets/ideval99/week2/tuesday/inside.tcpdump
Generated: 2013-09-24 15:32:50

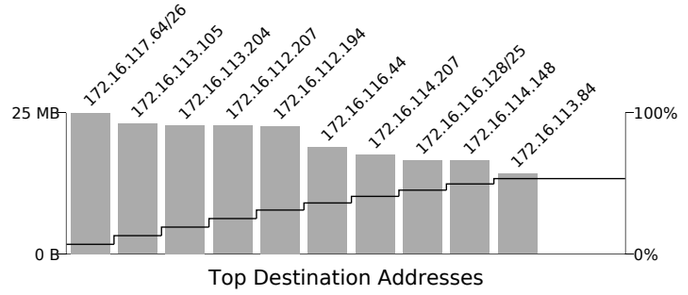
Date range: 1999-03-09 08:00:01 -- 1999-03-10 03:03:20
Packets analyzed: 1,571,748 (373.65 MB)
Transports: IPv4 100%





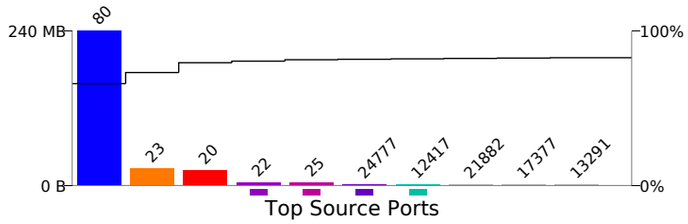
Top Source Addresses

1) 172.16.112.194 - 16.15 MB (4%)
2) 172.16.112.100 - 15.86 MB (4%)
3) 206.128.0.0/9 - 12.54 MB (3%)



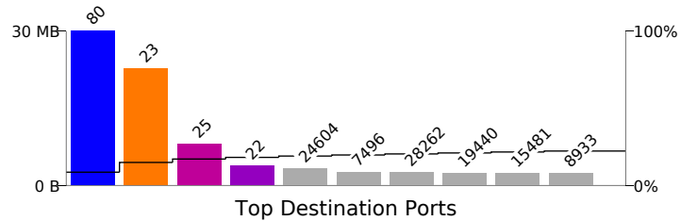
Top Destination Addresses

1) 172.16.117.64/26 - 24.91 MB (6%)
2) 172.16.113.105 - 22.98 MB (6%)
3) 172.16.113.204 - 22.69 MB (6%)



Top Source Ports

1) 80 - 239.63 MB (65%)
2) 23 - 26.32 MB (7%)
3) 20 - 23.17 MB (6%)



Top Destination Ports

1) 80 - 30.44 MB (8%)
2) 23 - 23.06 MB (6%)
3) 25 - 8.05 MB (2%)

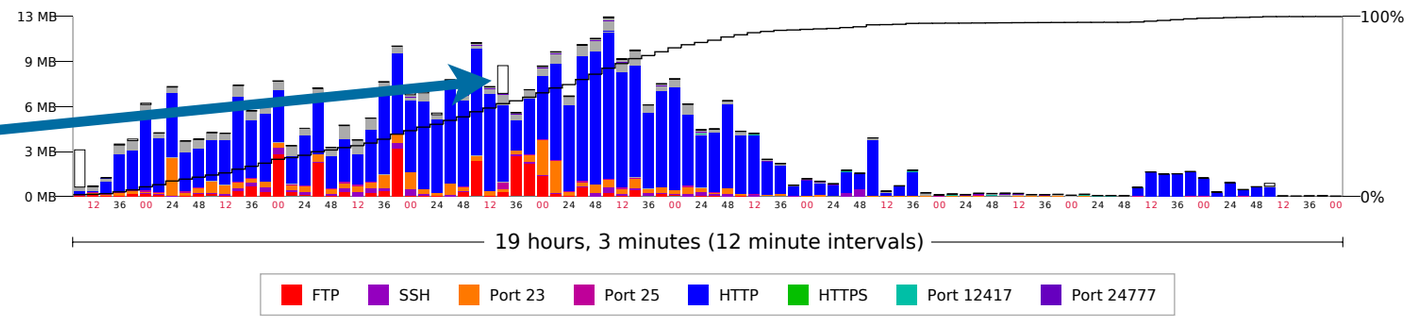
MIT ID'99 IDS evaluation.
The graph shows peculiarities of the traffic.

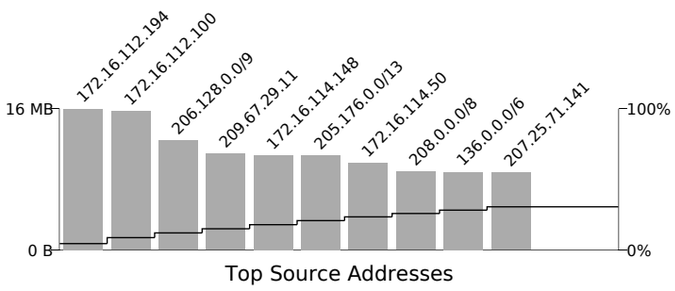
- Lots of FTP
- Not enough off-peak data.
- Lots of non-TCP

likely smurf attack

TCPFLOW 1.4.0
Input: /corp/mitll/packets/ideval99/week2/tuesday/inside.tcpdump
Generated: 2013-09-24 15:32:50

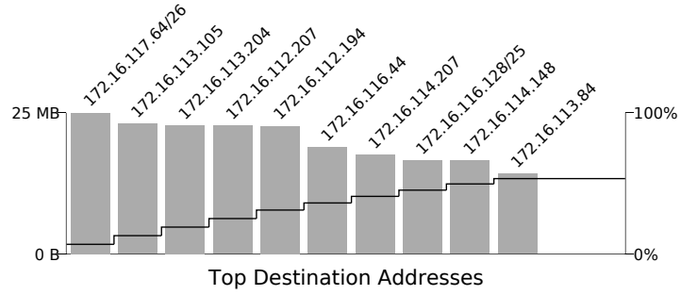
Date range: 1999-03-09 08:00:01 -- 1999-03-10 03:03:20
Packets analyzed: 1,571,748 (373.65 MB)
Transports: IPv4 100%





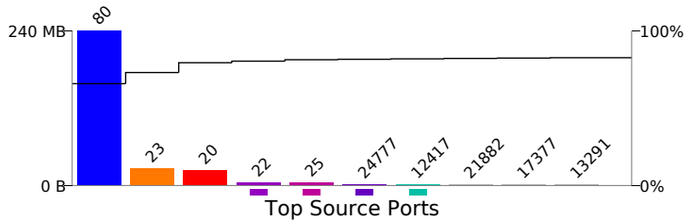
Top Source Addresses

1) 172.16.112.194 - 16.15 MB (4%)
2) 172.16.112.100 - 15.86 MB (4%)
3) 206.128.0.0/9 - 12.54 MB (3%)



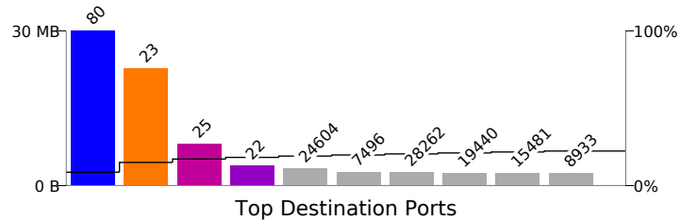
Top Destination Addresses

1) 172.16.117.64/26 - 24.91 MB (6%)
2) 172.16.113.105 - 22.98 MB (6%)
3) 172.16.113.204 - 22.69 MB (6%)



Top Source Ports

1) 80 - 239.63 MB (65%)
2) 23 - 26.32 MB (7%)
3) 20 - 23.17 MB (6%)



Top Destination Ports

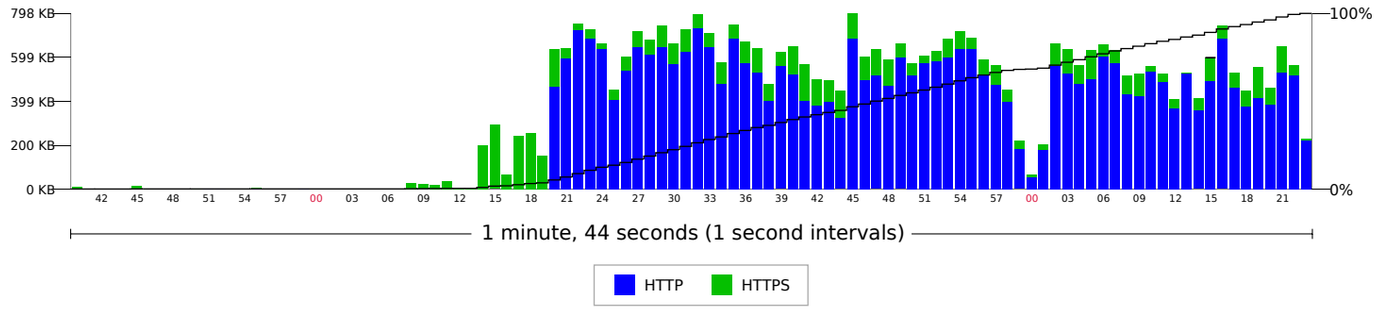
1) 80 - 30.44 MB (8%)
2) 23 - 23.06 MB (6%)
3) 25 - 8.05 MB (2%)

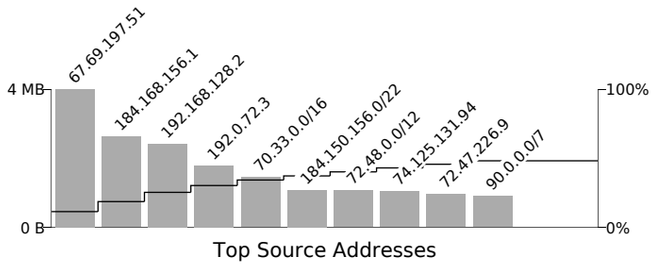
2 minutes of packets on a high-volume network

- Mostly HTTP & HTTPS
- Mostly to 192.168.128.2

TCPFLOW 1.4.0
Input: sun.pcap
Generated: 2013-09-05 20:35:09

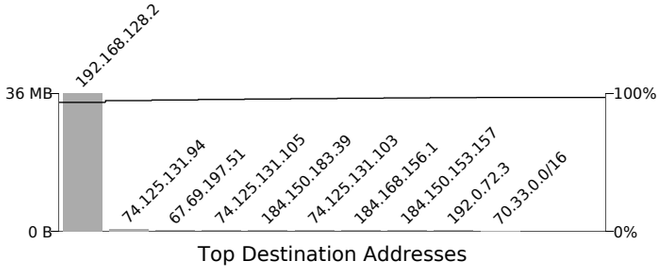
Date range: 2013-09-03 21:31:40 -- 2013-09-03 21:33:23
Packets analyzed: 62,088 (38.54 MB)
Transports: Other 100%





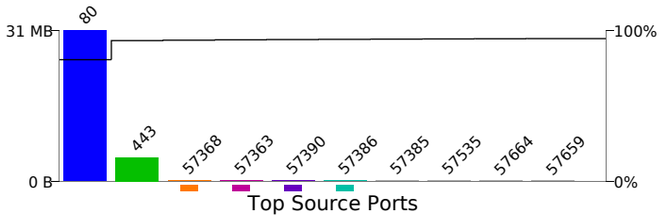
Top Source Addresses

1) 67.69.197.51 - 4.28 MB (11%)
2) 184.168.156.1 - 2.80 MB (7%)
3) 192.168.128.2 - 2.58 MB (6%)



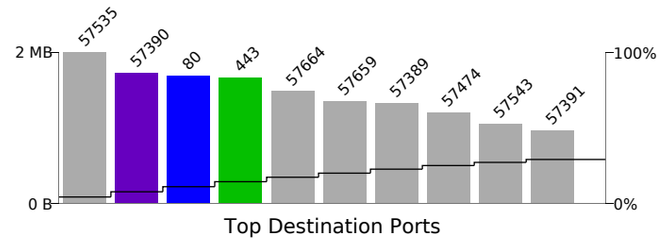
Top Destination Addresses

1) 192.168.128.2 - 35.95 MB (93%)
2) 74.125.131.94 - 493.82 KB (1%)
3) 67.69.197.51 - 172.48 KB (0%)



Top Source Ports

1) 80 - 31.07 MB (80%)
2) 443 - 4.89 MB (12%)
3) 57368 - 108.75 KB (0%)



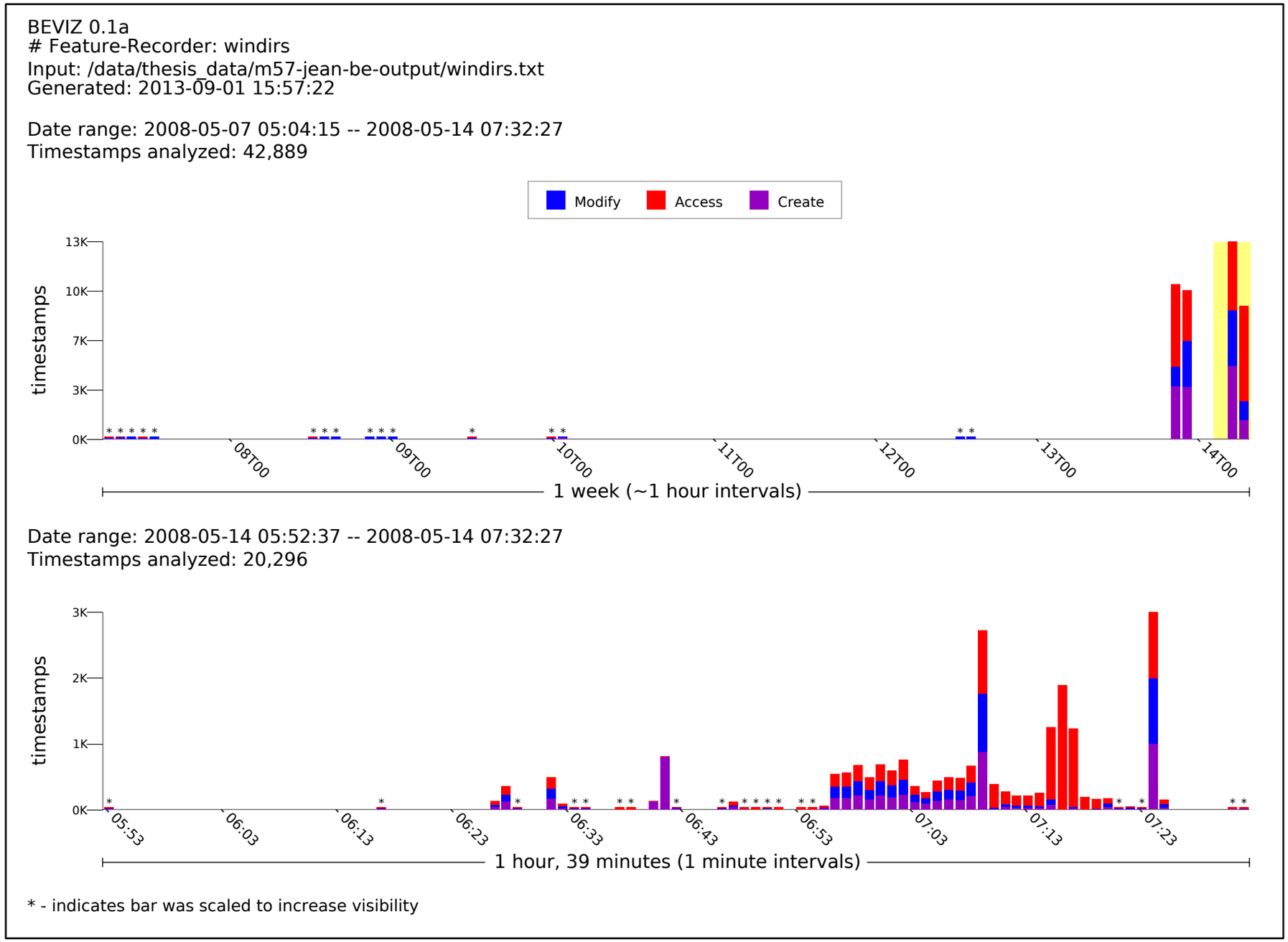
Top Destination Ports

1) 57535 - 1.55 MB (4%)
2) 57390 - 1.33 MB (3%)
3) 80 - 1.30 MB (3%)

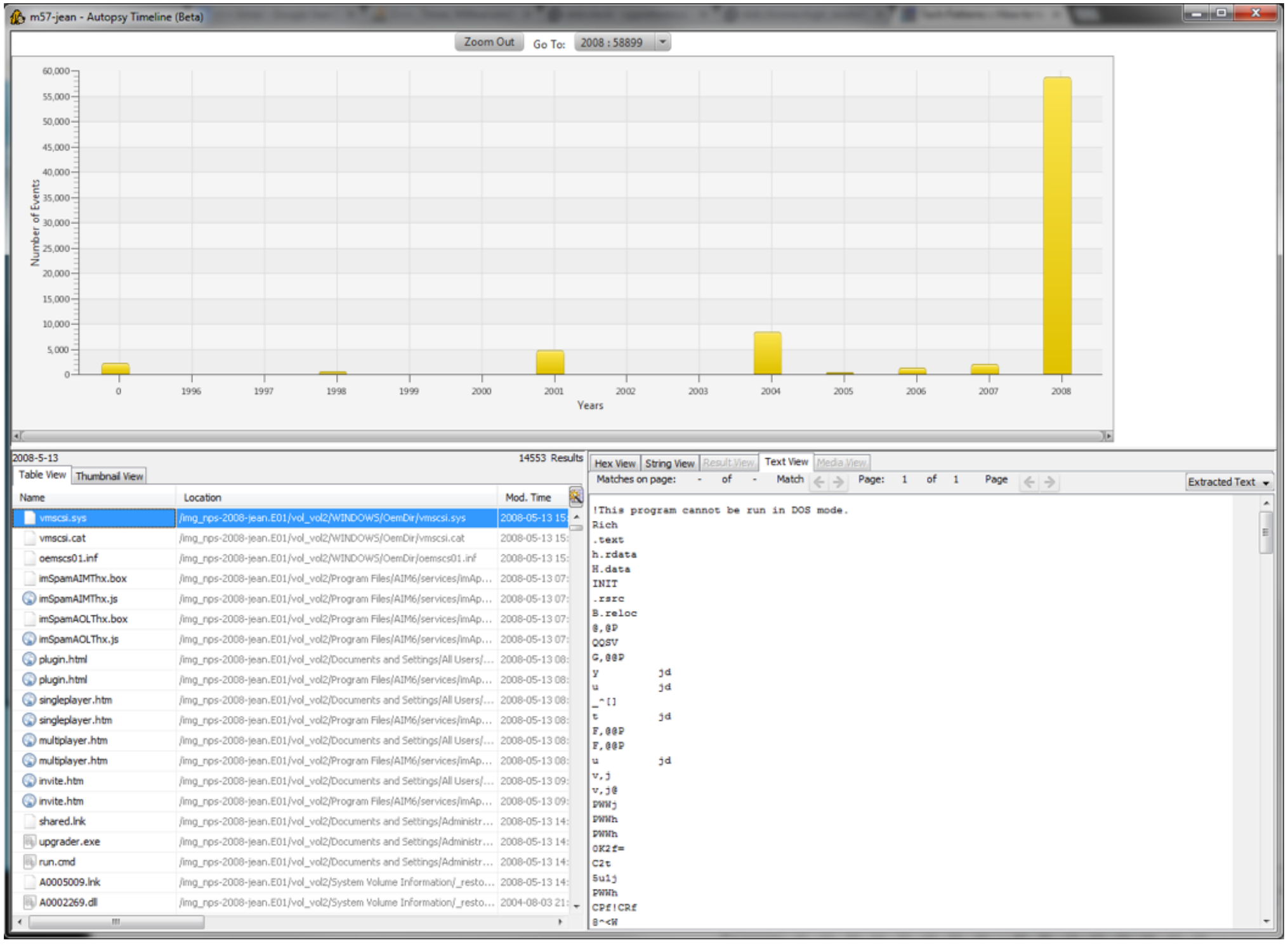
Glenn Henderson's MS thesis (NPS Sept 2013) applied this visualization to disk images (BE windirs.txt)

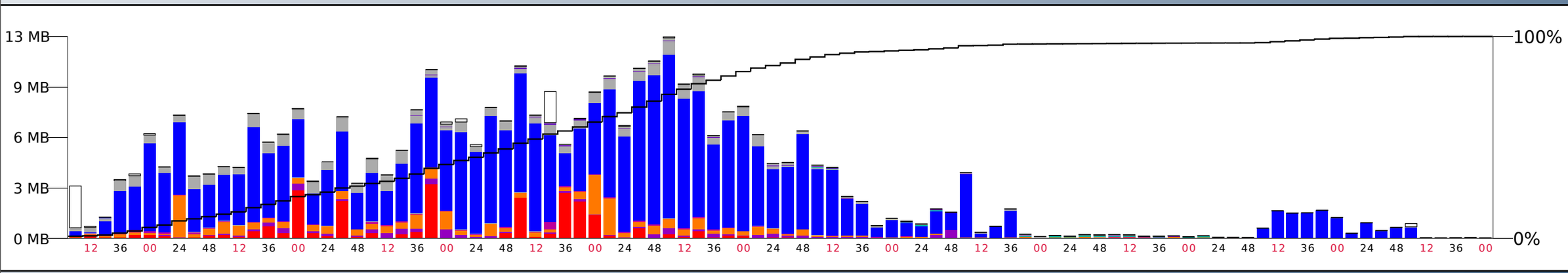
Key improvements:

- Zooms area of interest.
- Improved legends.



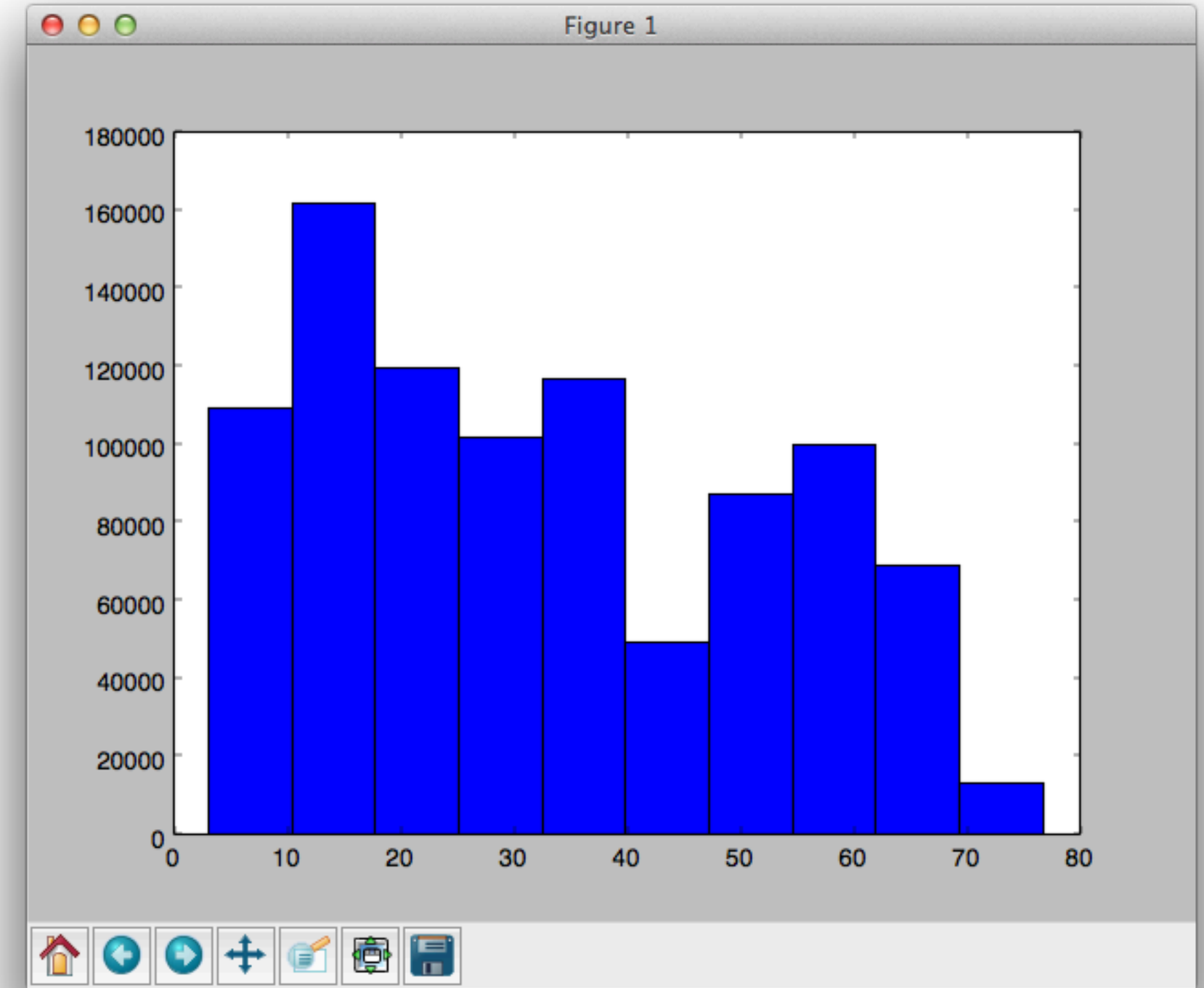
Same disk image viewed with Autopsy Timeline (beta)





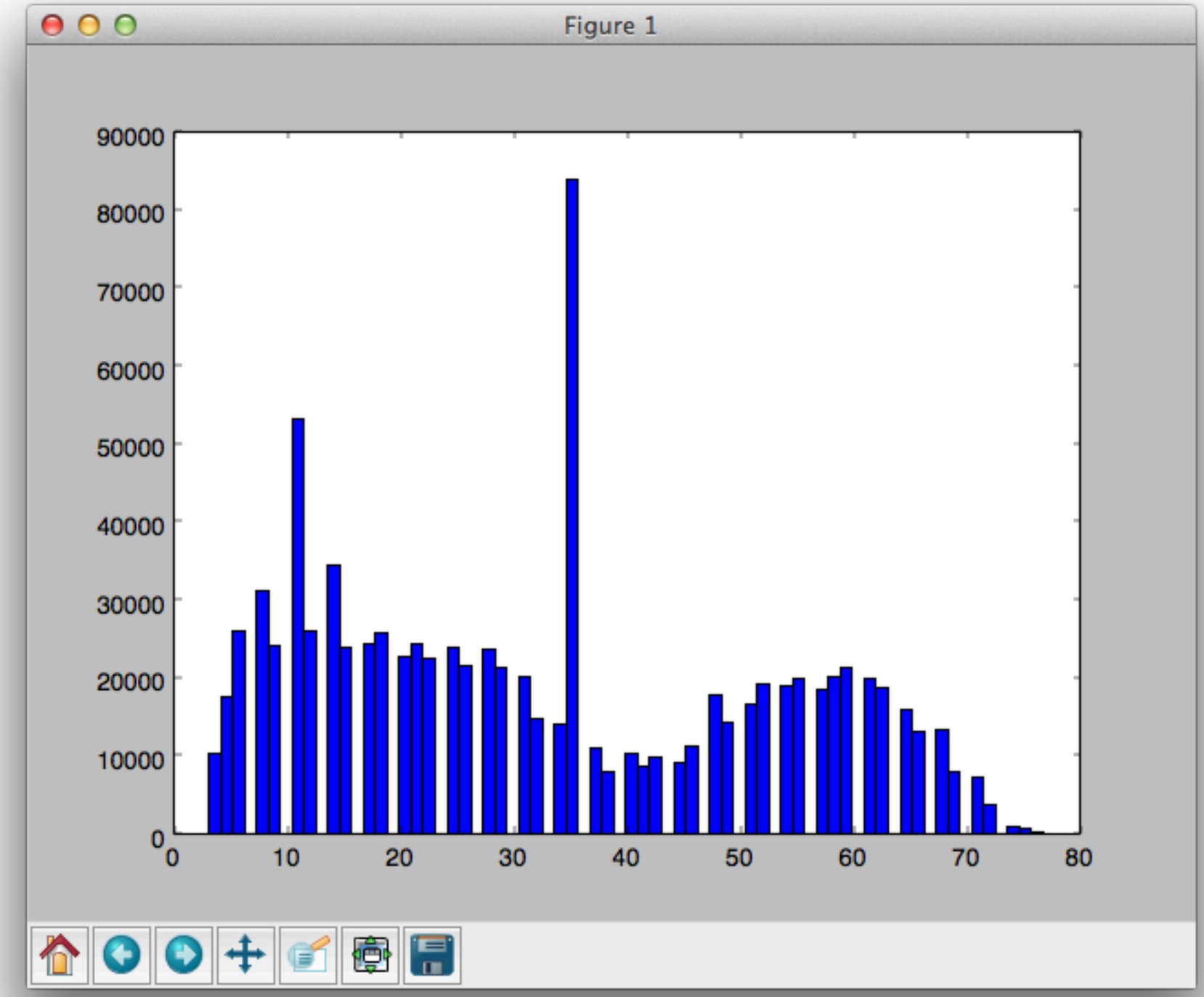
Histograms vs. CDFs

Problem with histograms: # of bins changes the results.

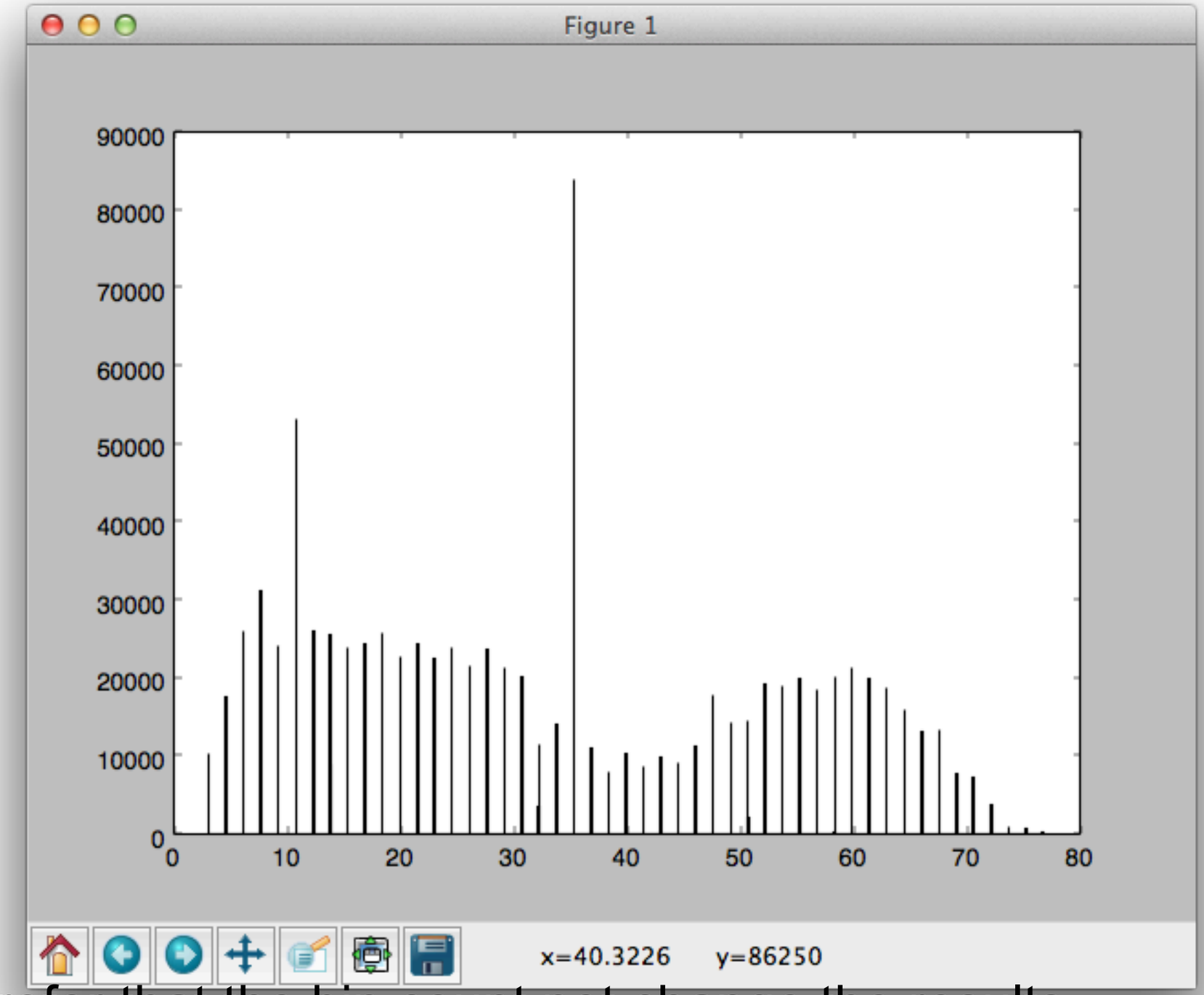


10 bins

Histogram with 70 bins



Histogram with 1000 bins:

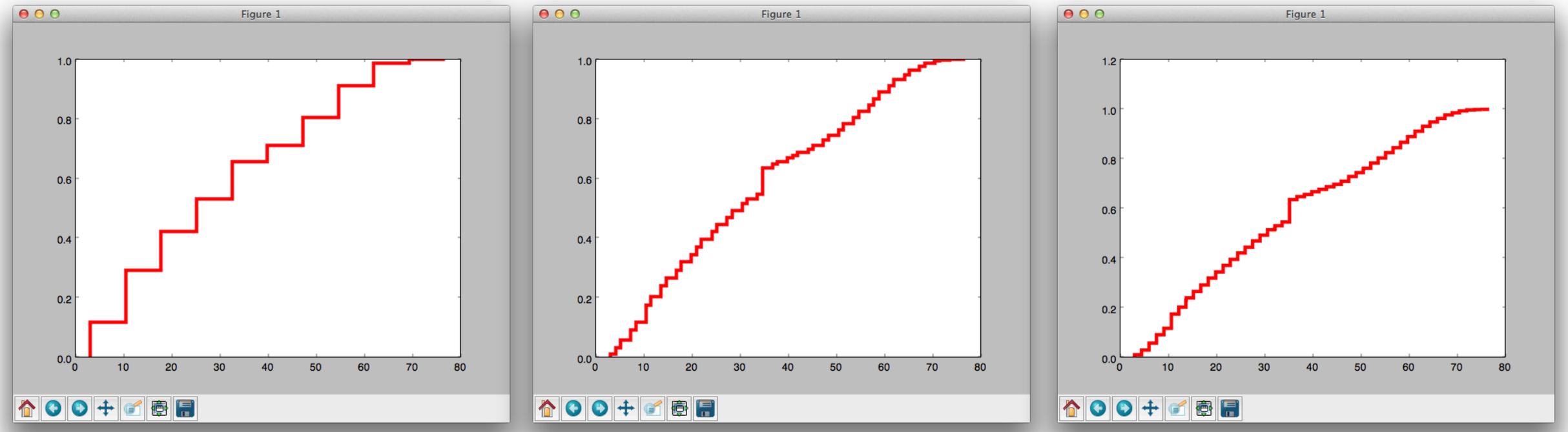


We would prefer that the bin count not change the results.

A cumulative distribution function (CDF) plot is less sensitive to bin count.

A CDF shows the fraction of measurements less than a value.

Many people find CDFs hard to understand.

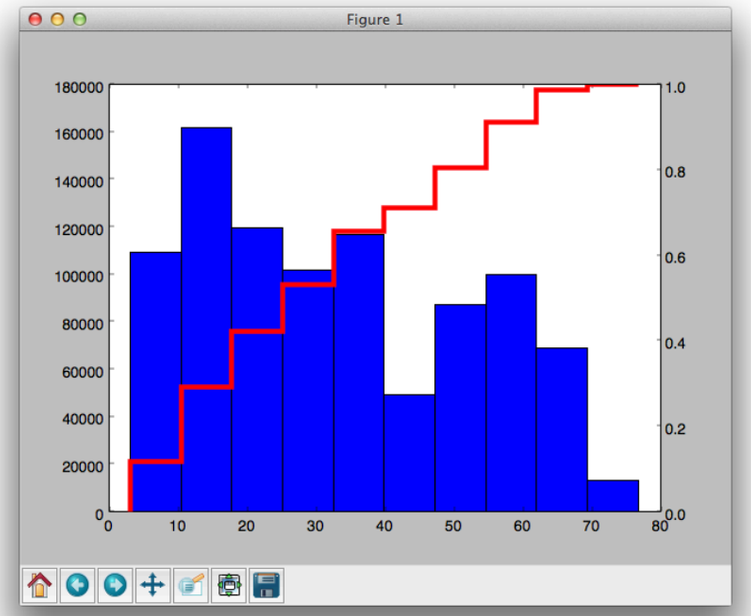


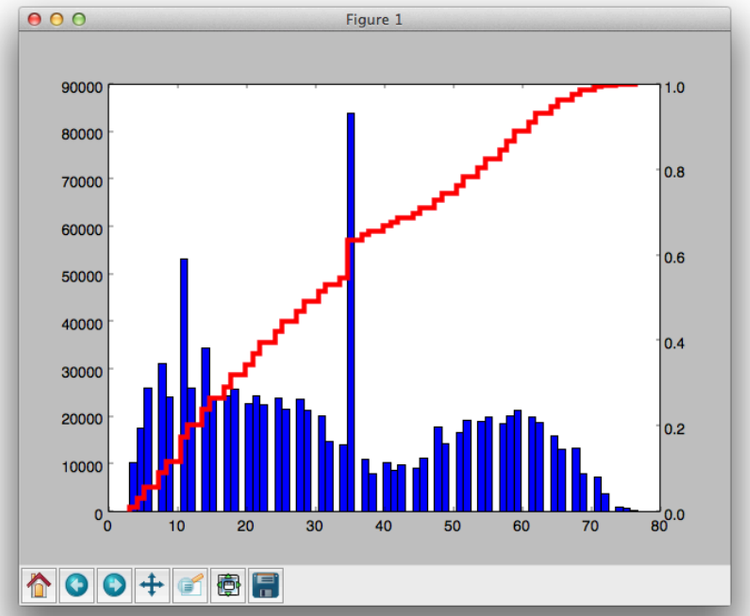
bins=10 **bins=70** **bins=1000**

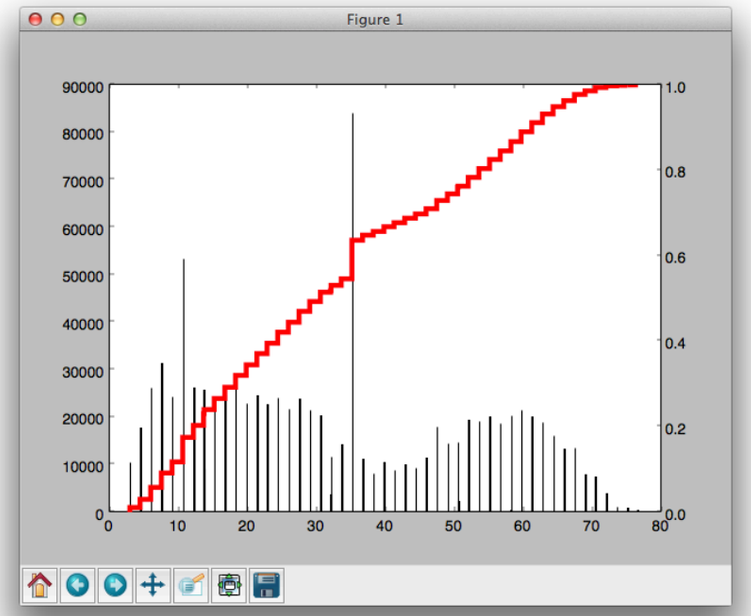
CDFs are easy to make with matplotlib:

```
P.hist(speed_vals,bins,weights=speed_secs,cumulative=True,
       histtype='step',normed=True,color='red',linewidth=4)
```

I like overlaying the CDF on a histogram. This is *easy* to do with matplotlib



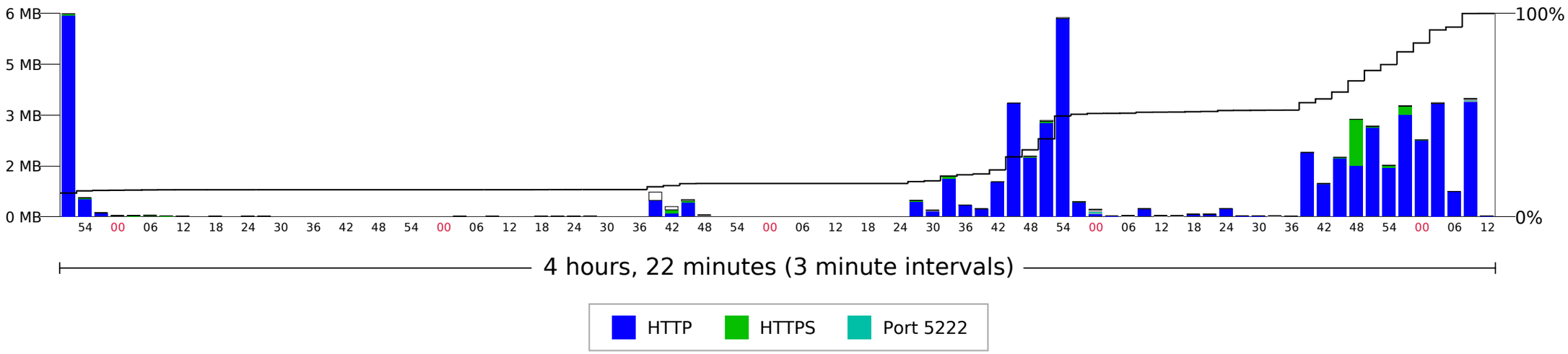


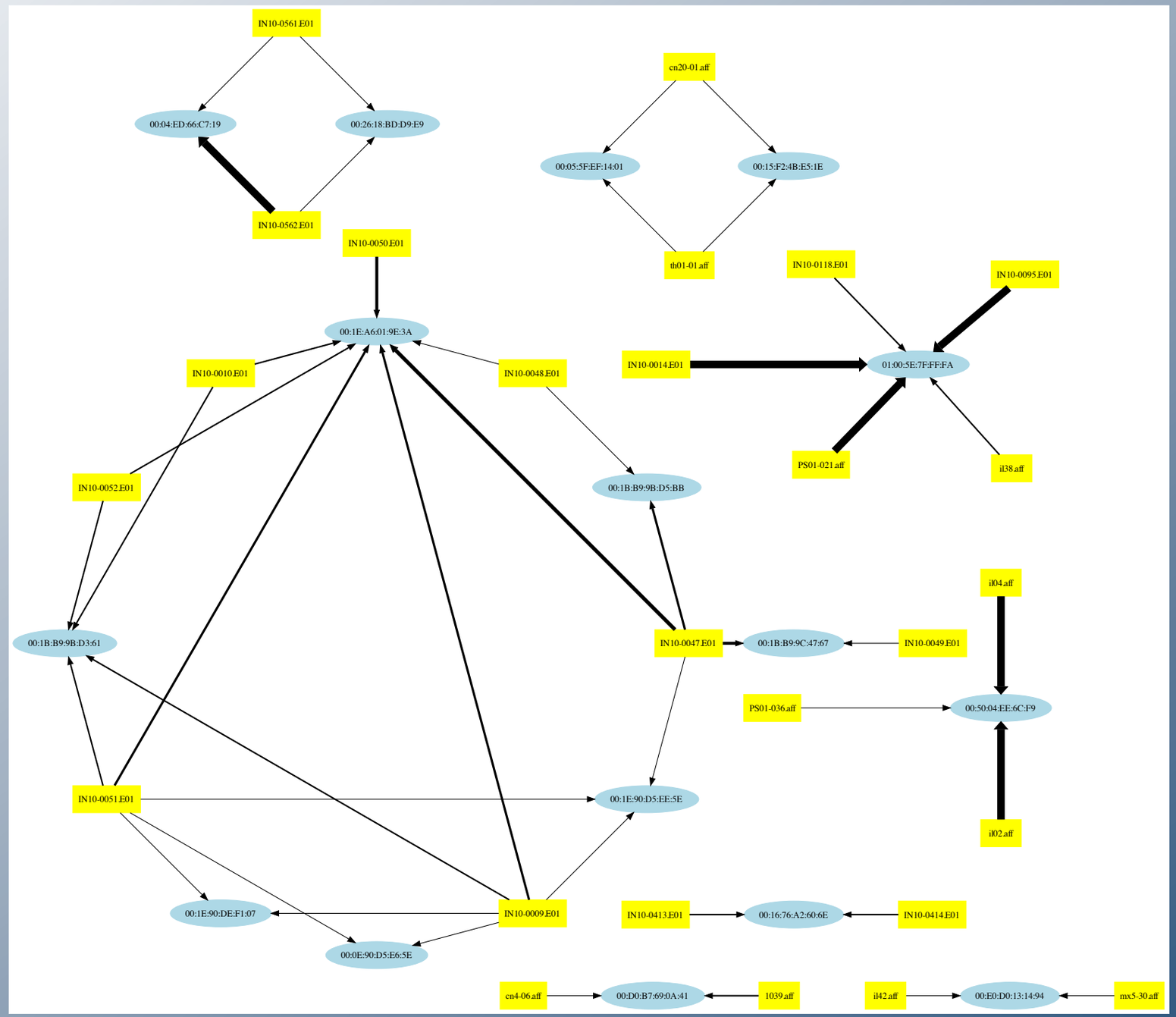


bins=10 **bins=70** **bins=1000**

```
fig, ax1 = plt.subplots()

ax1.hist(speed_vals,bins,weights=speed_secs,log=False)

ax2 = ax1.twinx()
ax2.set_ylim([0,1])
ax2.hist(speed_vals,bins,weights=speed_secs,cumulative=True,
         histtype='step',normed=True,color='red',linewidth=4)
plt.show()
```

The overlay on tcpflow is subtle.
We hope people can figure it out (but we haven't tested).





Network Visualization

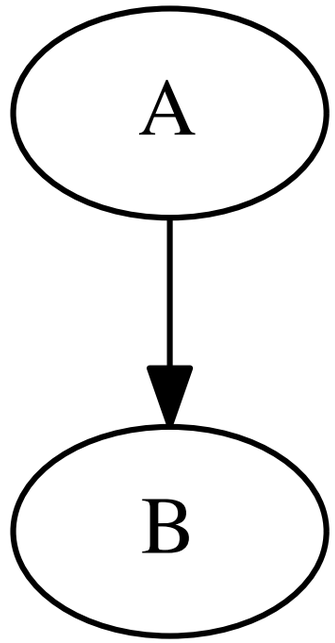
GraphViz is an easy tool for network visualization

Originally developed by Bell Labs

Multiple layout engines

Simple “language” for describing graphs

```
digraph G {
        A -> B;
}

$ dot -Tpdf demo1.dot -o demo1.pdf
```

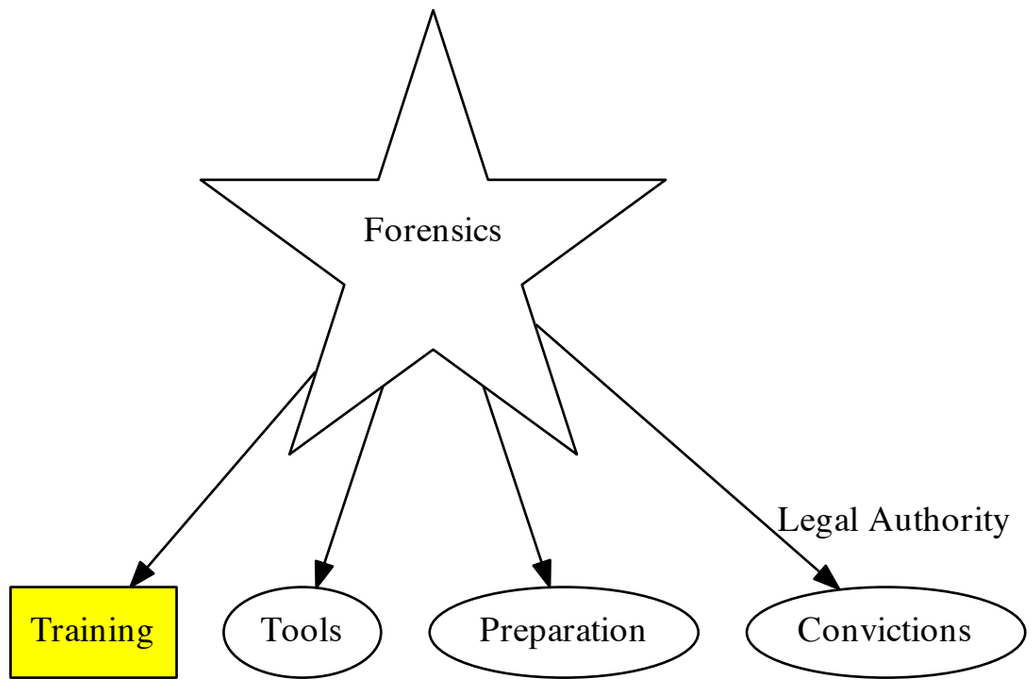


GraphViz allows tremendous flexibility

You can change:

- Object shapes, colors
- Layout algorithm

```
digraph G {
        C [shape=star,label="Forensics",height=2];
        Training [shape=box,style=filled,fillcolor=yellow];
        C -> Training;
        C -> Tools;
        C -> Preparation;
        C -> Convictions [label="Legal Authority"];
}
```



Graphviz offers several different layout engines. Layout is hard — especially for forensic data

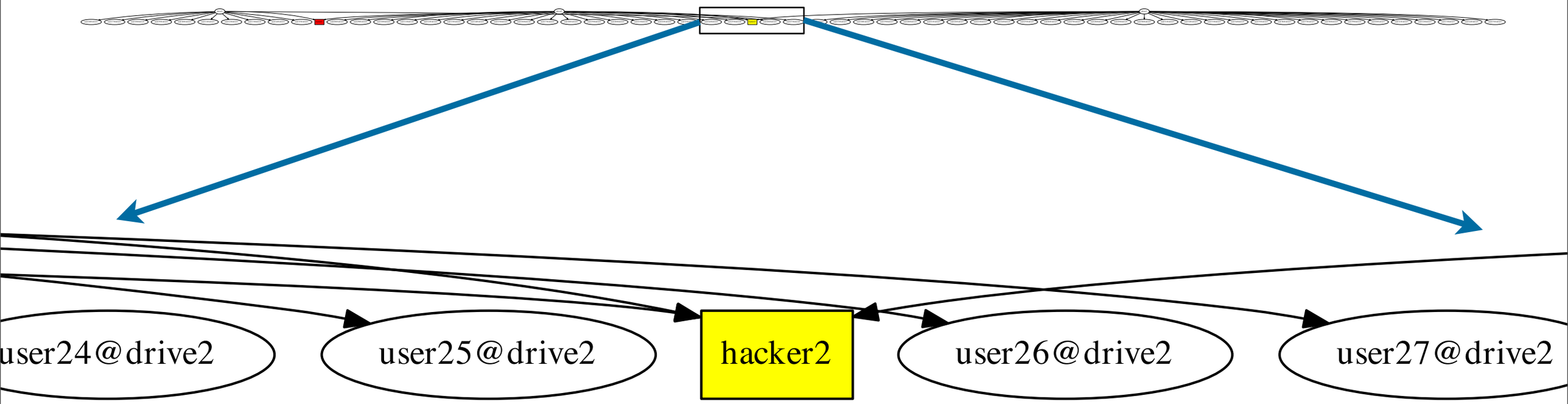
We have a *lot* of extraneous information

Consider a hypothetical case:

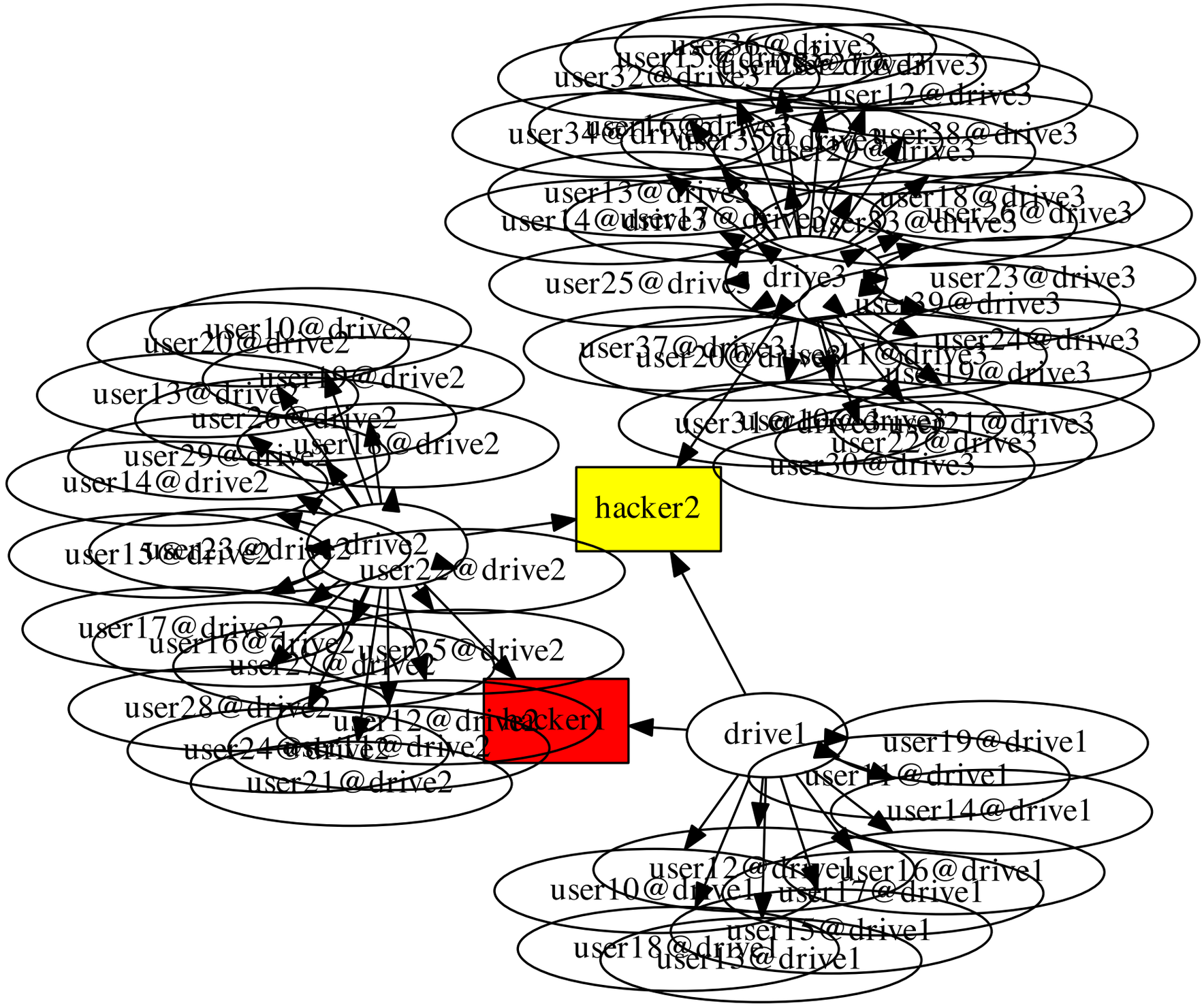
- drive #1 — 10 distinct email addresses
- drive #2 — 20 distinct email addresses
- drive #3 — 30 distinct email addresses
- hacker1 — common between drive #1, #2
- hacker2 — common between drive #1, #2, #3

```
drive1_emails = ["user%d@drive1" % i for i in range(10,20)]
drive2_emails = ["user%d@drive2" % i for i in range(10,30)]
drive3_emails = ["user%d@drive3" % i for i in range(10,40)]

for drive in [drive1_emails,drive2_emails]:
    drive += ["hacker1"]

for drive in [drive1_emails,drive2_emails,drive3_emails]:
    drive += ["hacker2"]
```

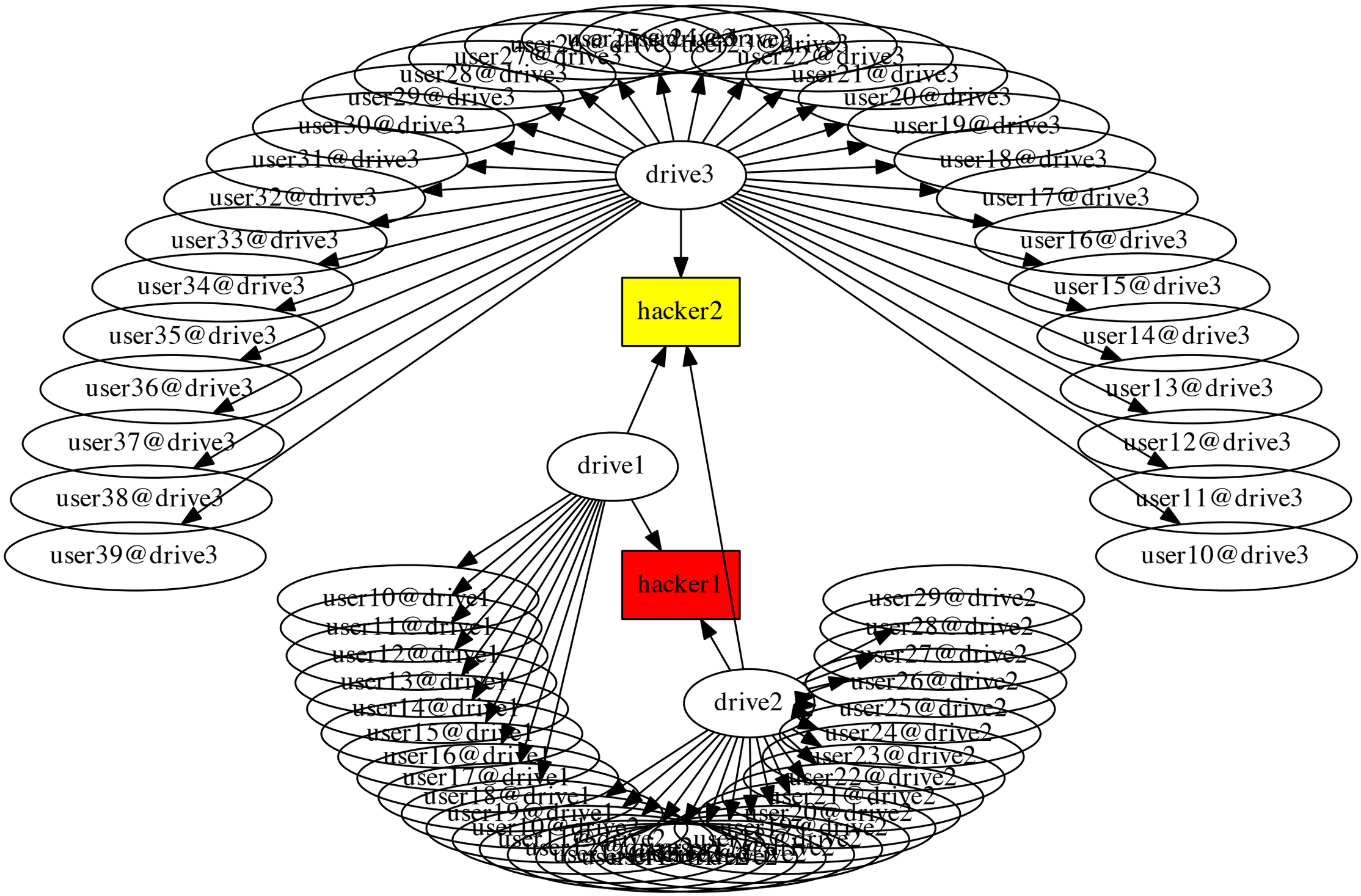
$ dot -Tpdf emailgraph.dot -o emailgraph-dot.pdf
(filter for drawing directed graphs; best with hierarchies.)



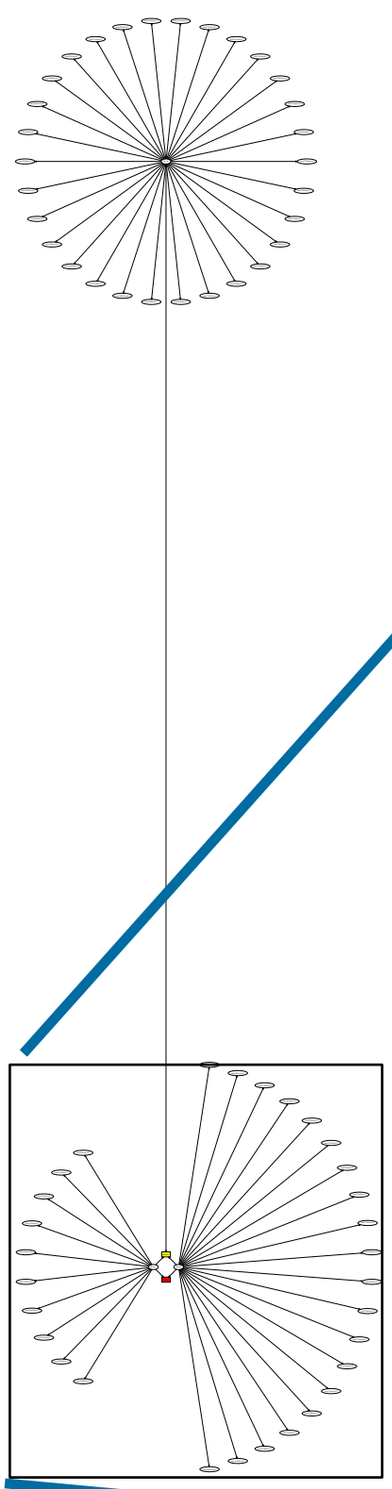
$ neato -Tpdf emailgraph.dot -o emailgraph-neato.pdf
Filter for drawing undirected graphs using spring models.

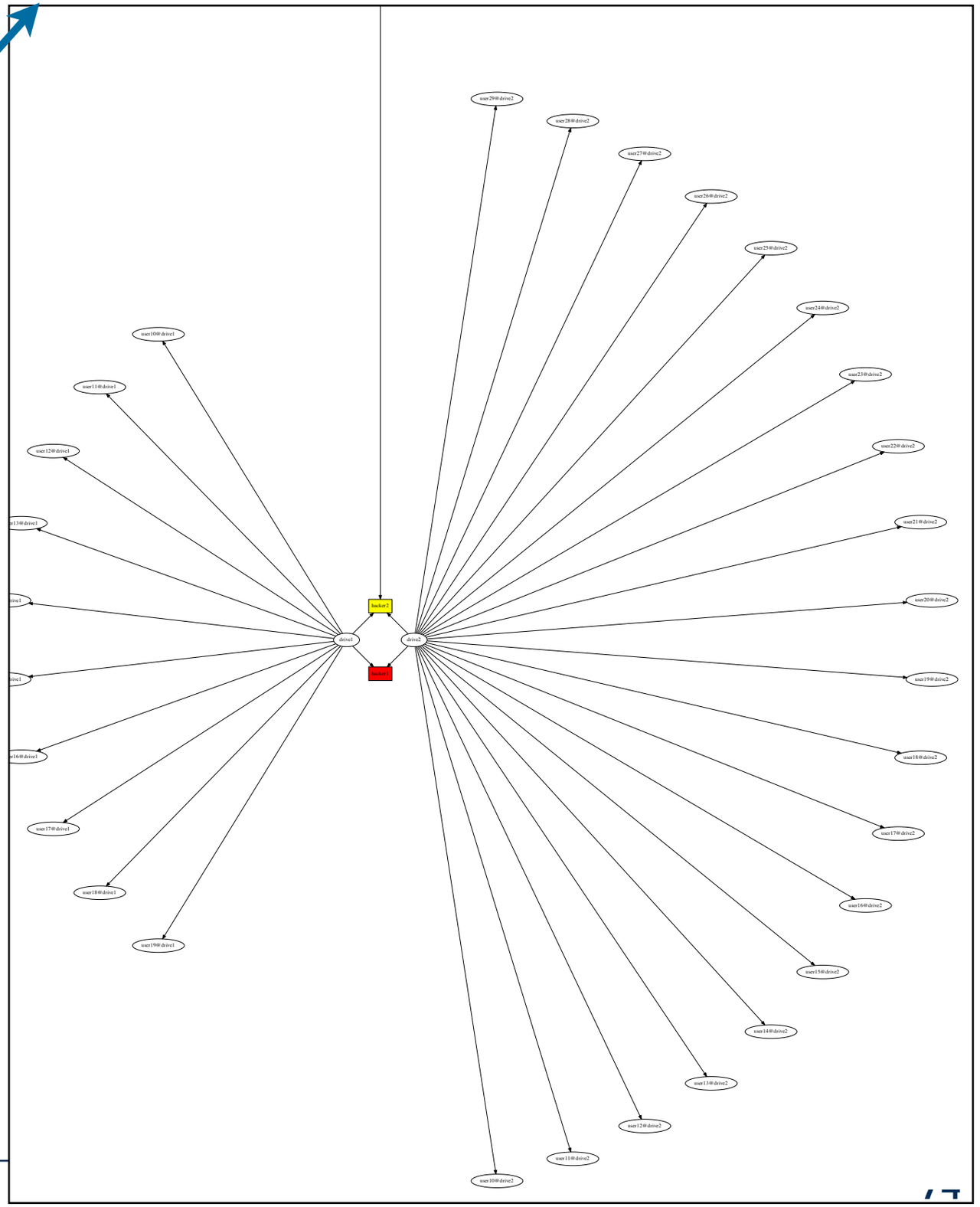


$ twopi -Tpdf emailgraph.dot -o emailgraph-twopi.pdf (Radial layout engine.)

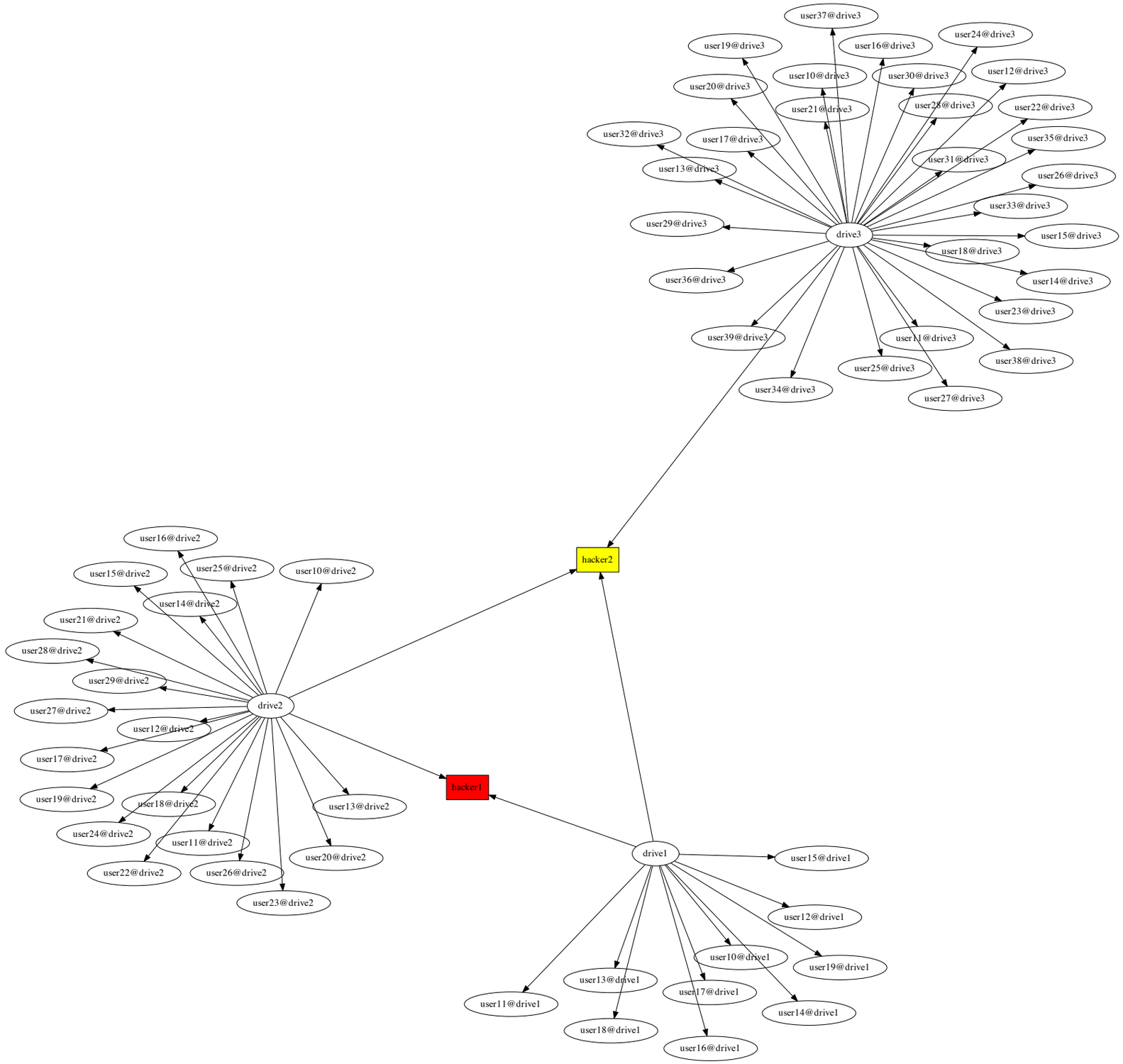


$ circio -Tpdf emailgraph.dot -o emailgraph-twopi.pdf (Circular layout engine.)

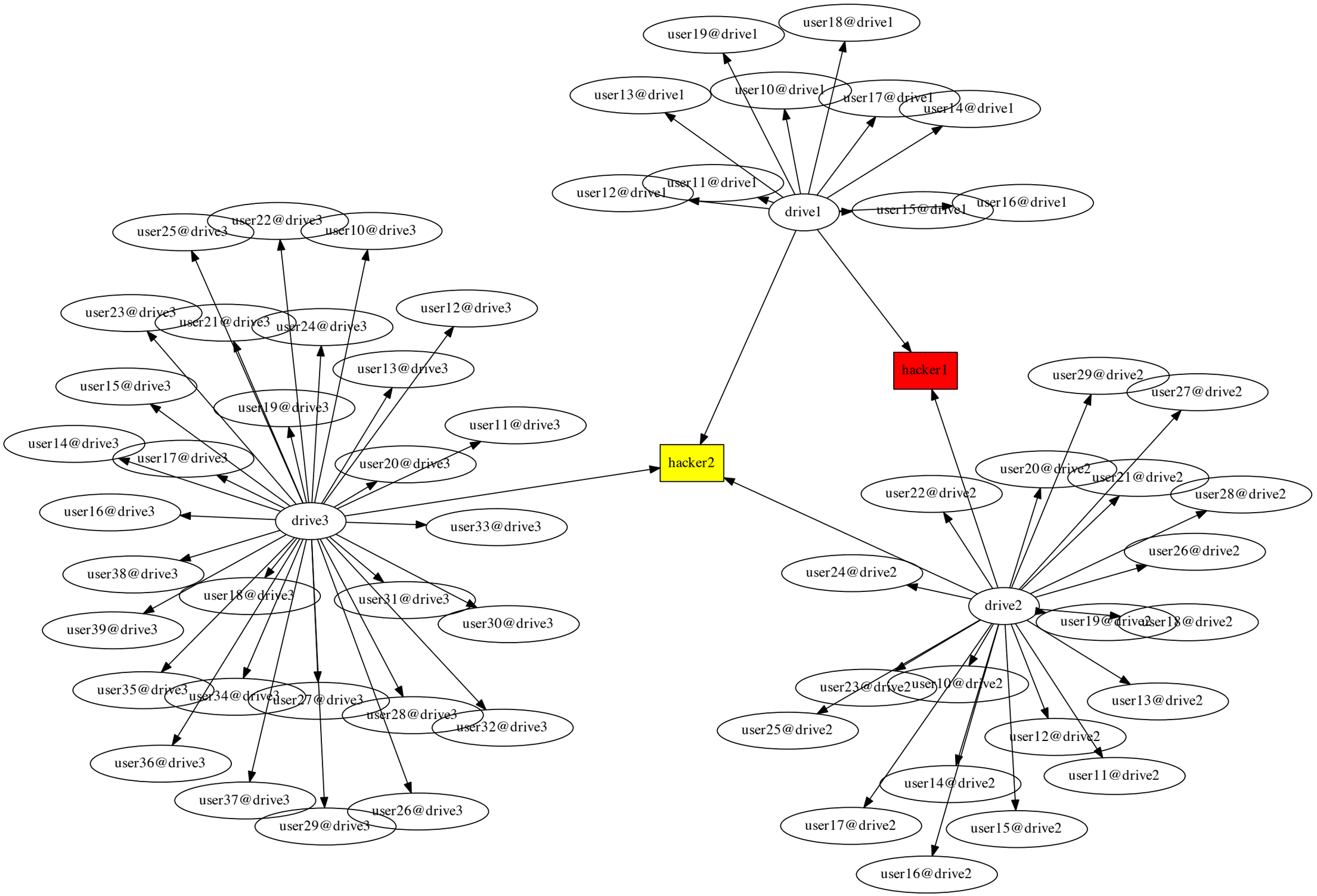




$ fdp -Tpdf emailgraph.dot -o emailgraph-fdp.pdf (Undirected graphs using “spring model.”)

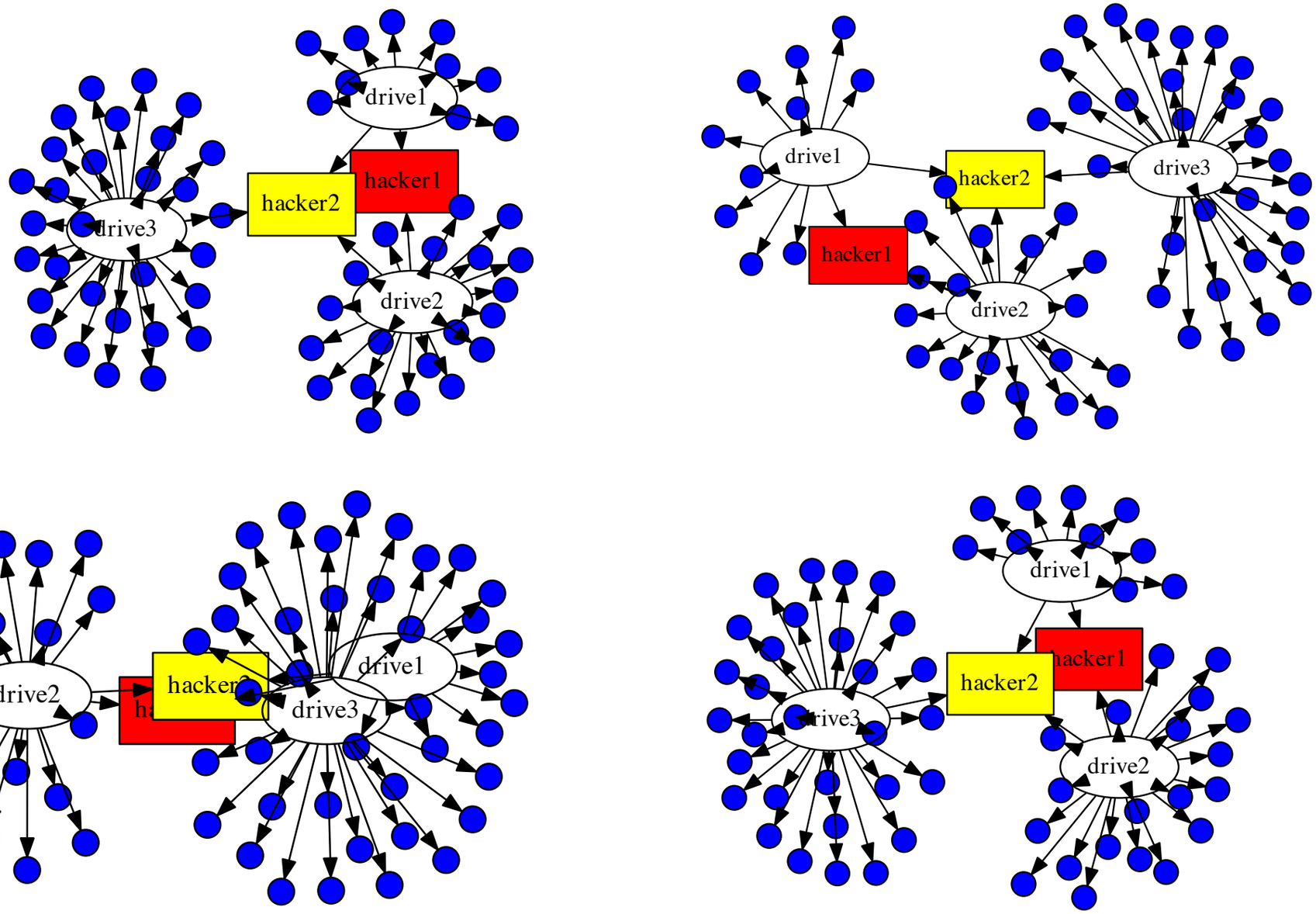


$ sfdp -Tpdf emailgraph.dot -o emailgraph-sfdp.pdf (Large undirected graphs with spring model.)



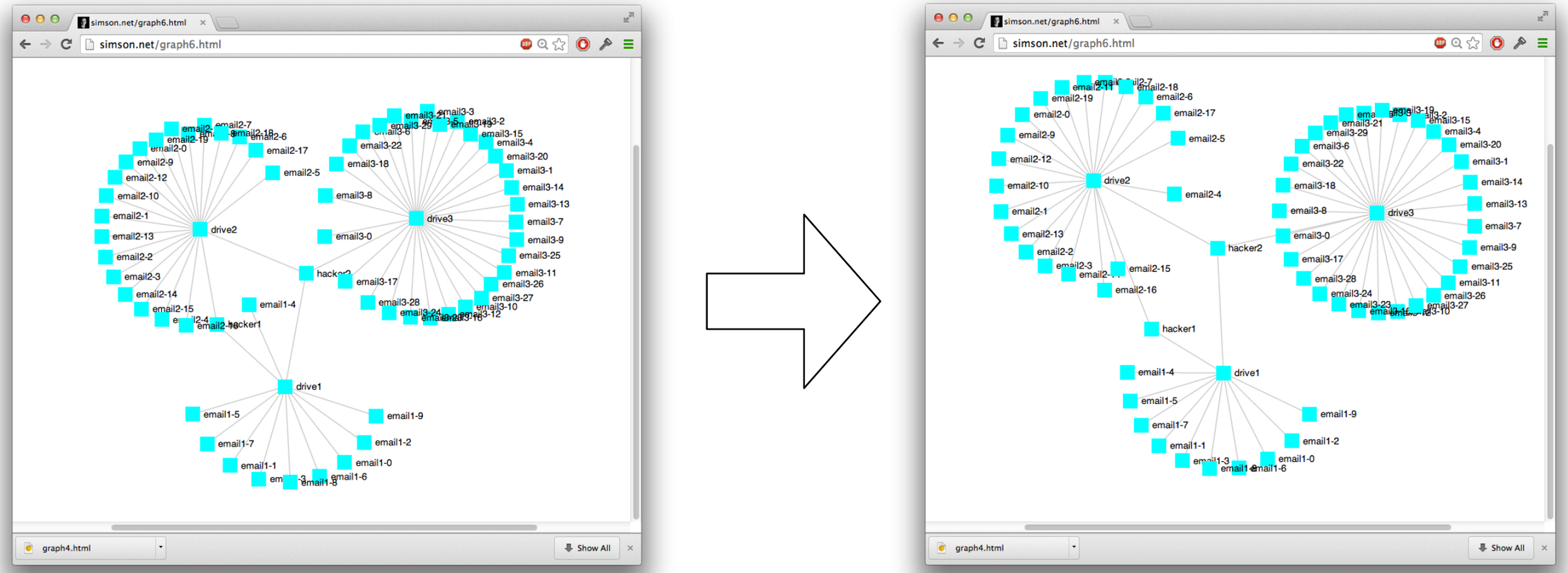
Improve the graph by *removing* information.
$ sfdp -Tpdf emailgraph.dot -o emailgraph-sfdp.pdf

- Remove emails from nodes that do not connect
- sfdp is non-deterministic — four runs, four different graphs:



I visualized the same dataset with 3DJS:

Because 3DJS is interactive, you can “fix” the layout.



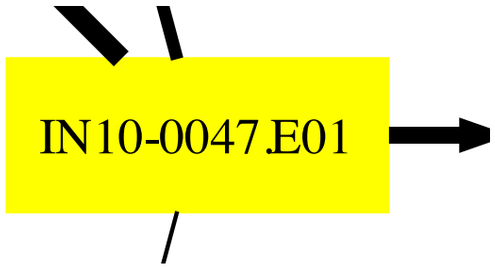
The only way to save this is printing to PDF and screen capture.

- Doesn’t work well for very complex datasets

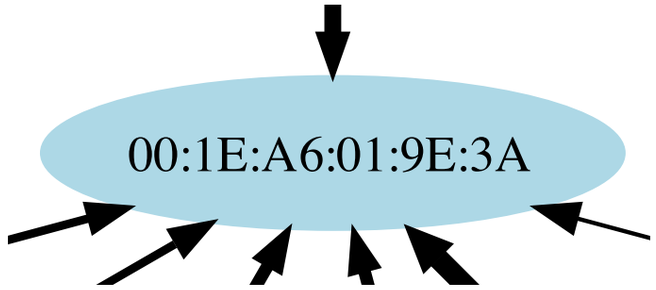
The IP carving “drives” visualization — A small number of important connections.

Nodes:

- Hard drives:



- MAC addresses:



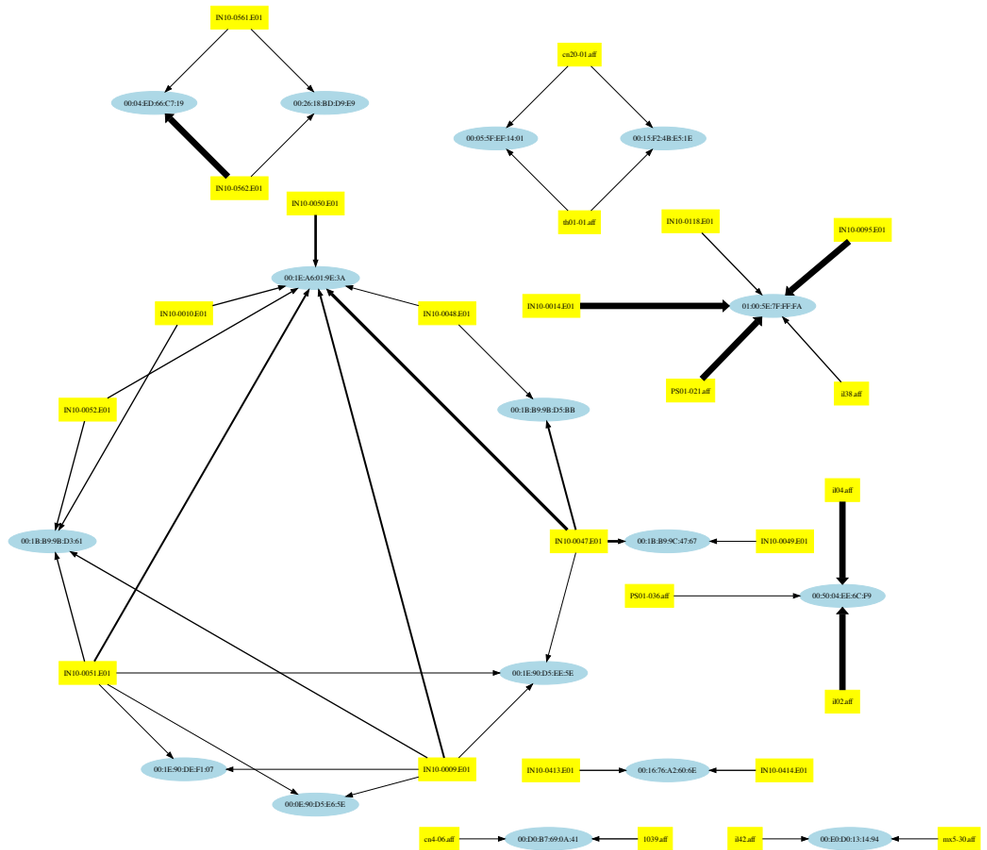
Edges:

- {drive} -> {MAC}

  —*when {MAC} found on {drive}*

Selection:

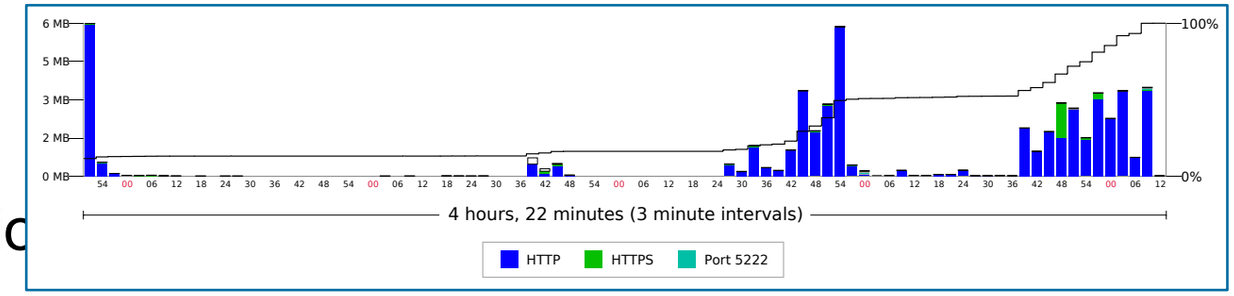
- {MAC} : only if on more than on {drive}
- {drive} : only if linked to a selected {MAC}



Conclusion: Data-driven visualization are an important growth area for open source forensics.

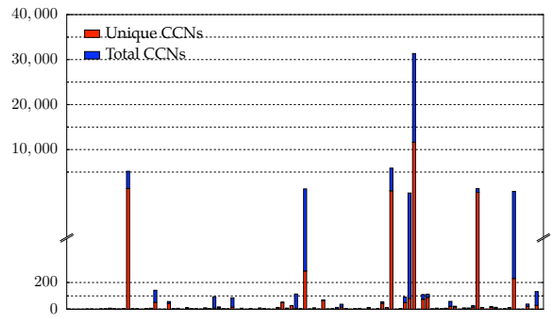
Specific requirements for forensic visualization:

- Repeatability
- No requirement for manual editing
- Ingests large amount of data
- PDF output



We need a “vocabulary” of forensic visualizatio

- Histograms w/ CDF overlay.
- Bar graphs
  - —*total vs. distinct.*
  - —*identify outliers with split axis vs. logarithmic*



Useful tools:

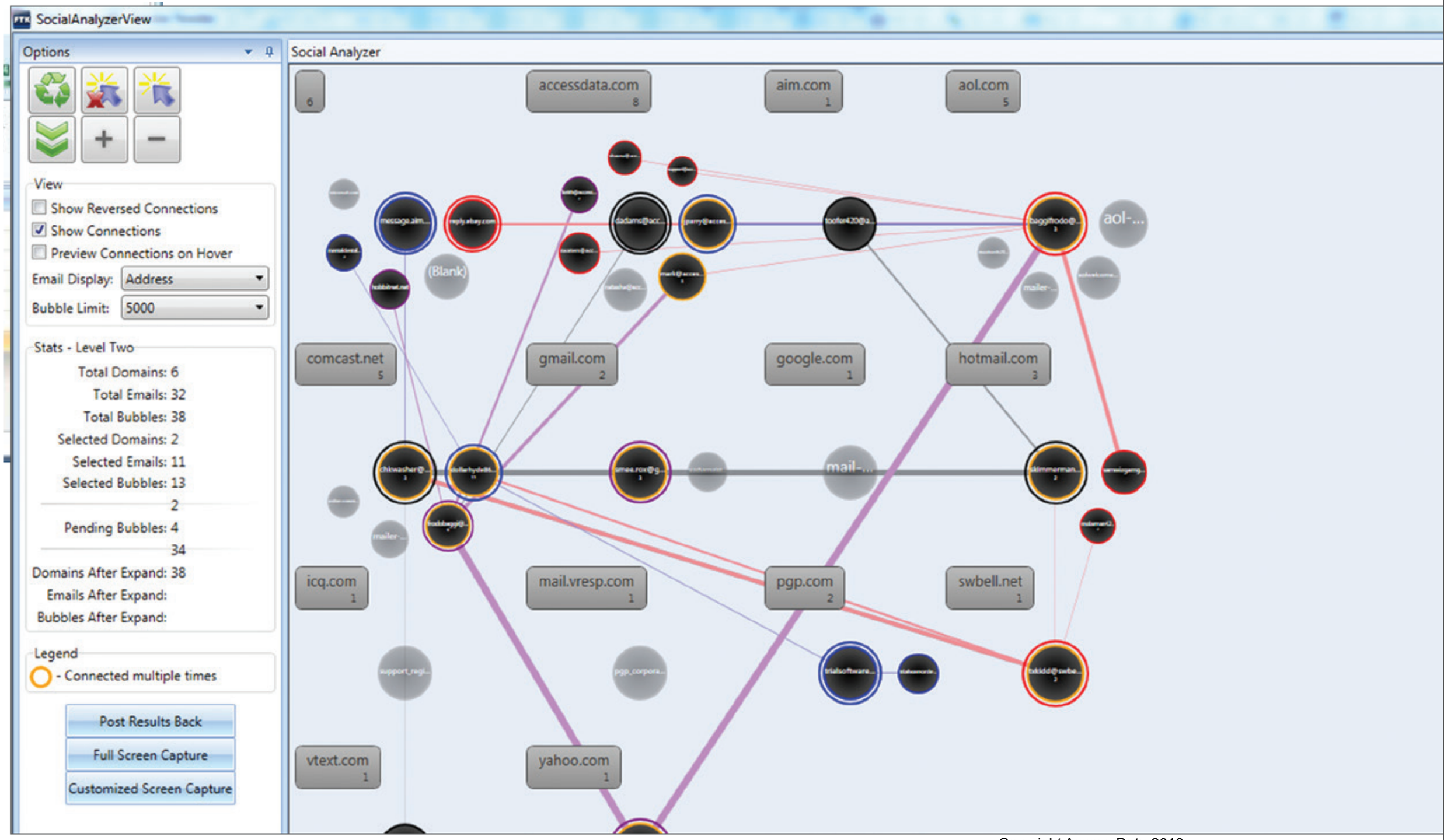
- matplotlib
- graphiviz
- 3DJS — Requires a browser

The role of browser-based visualization are unclear.

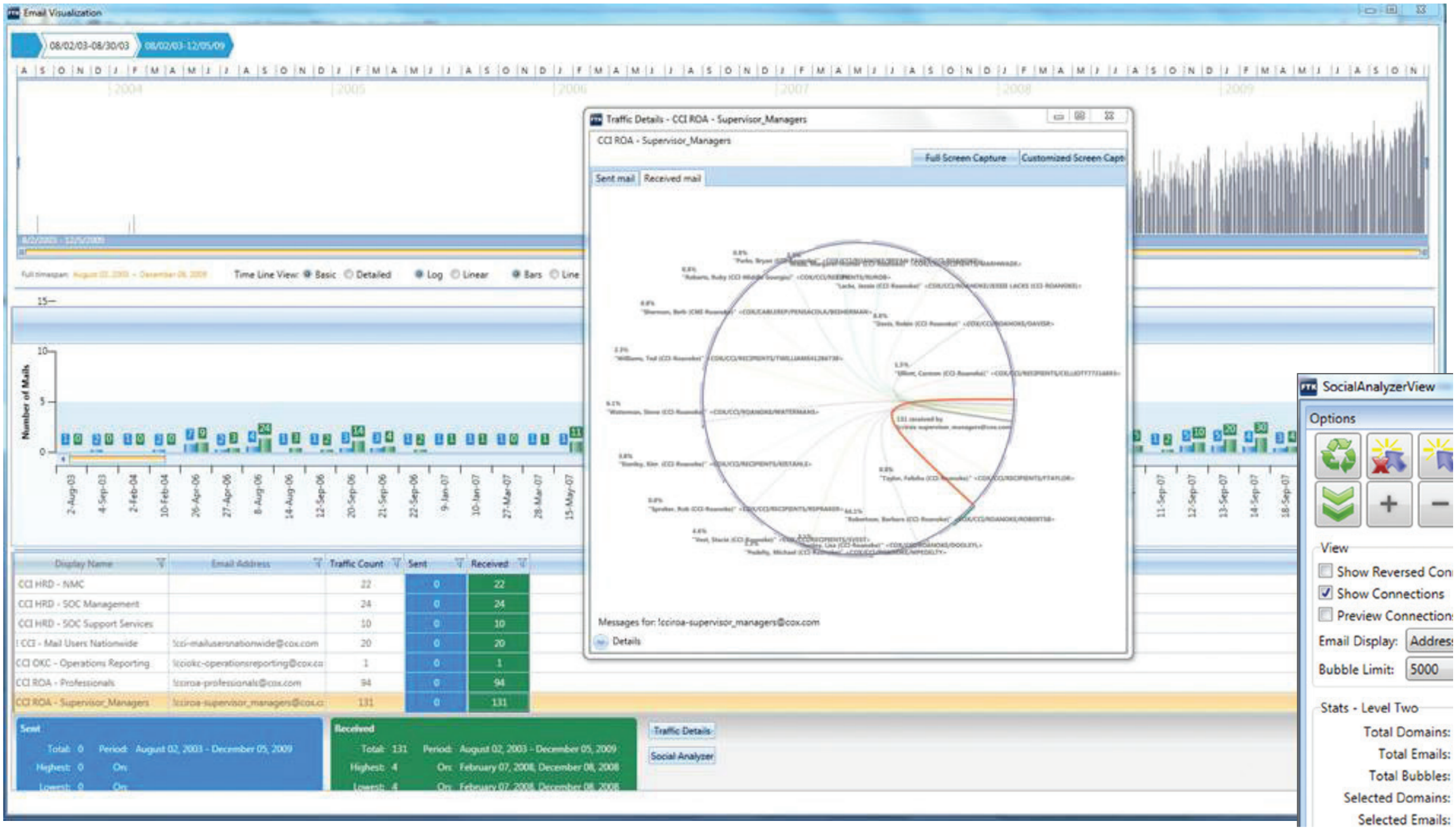
- Lots of good technology, but it may not fit our workflow.

Backup Slides

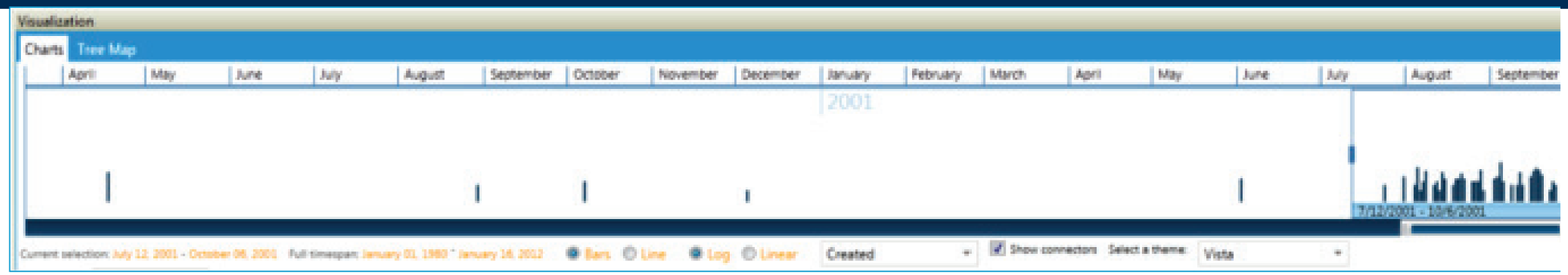
FTK5 (1/4)

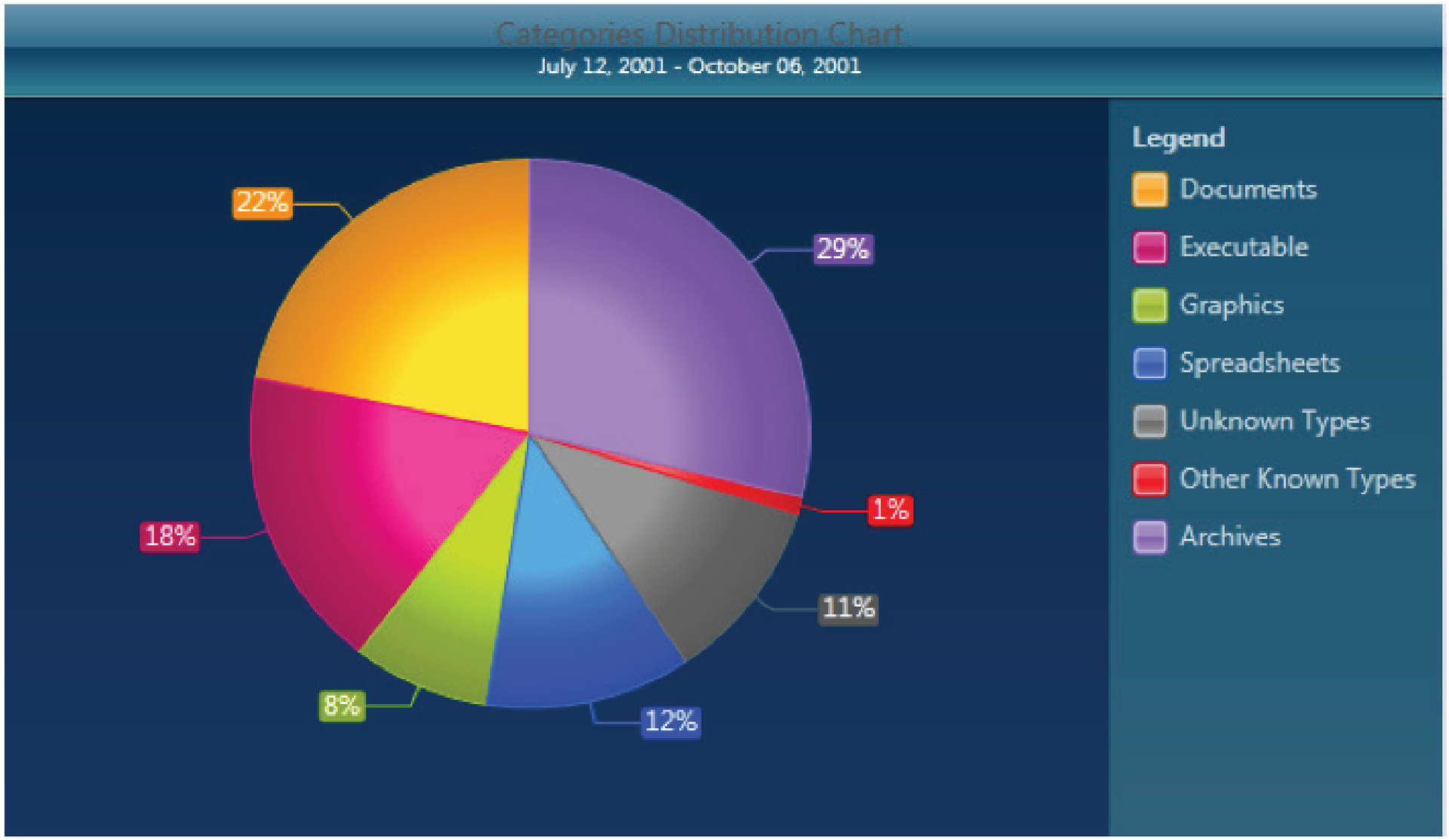


FTK5 (2/4)

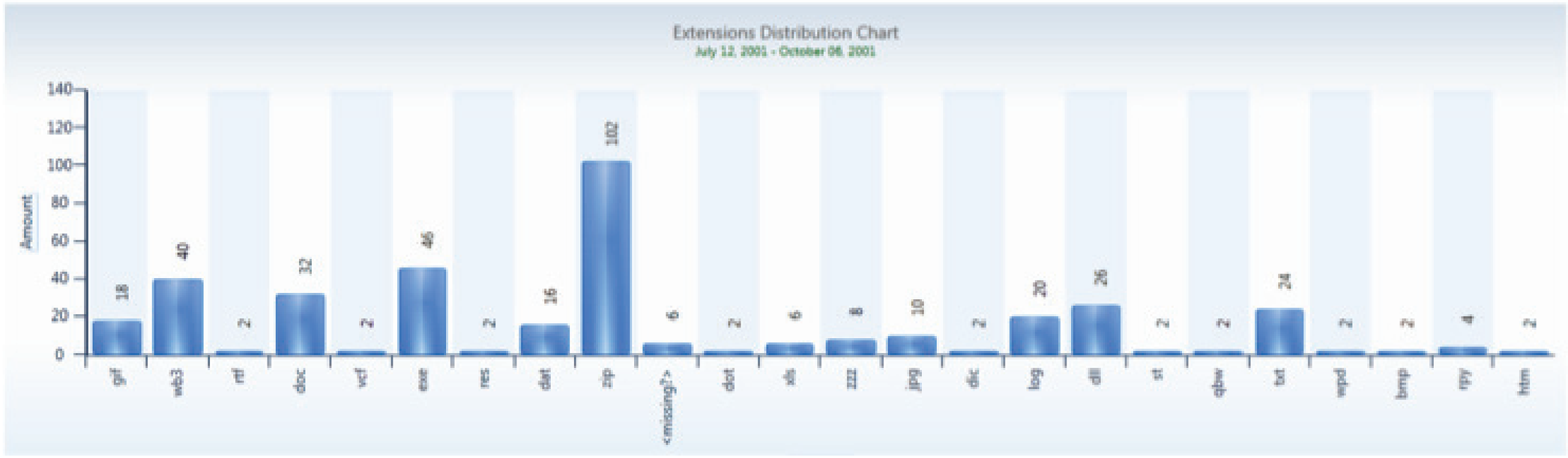


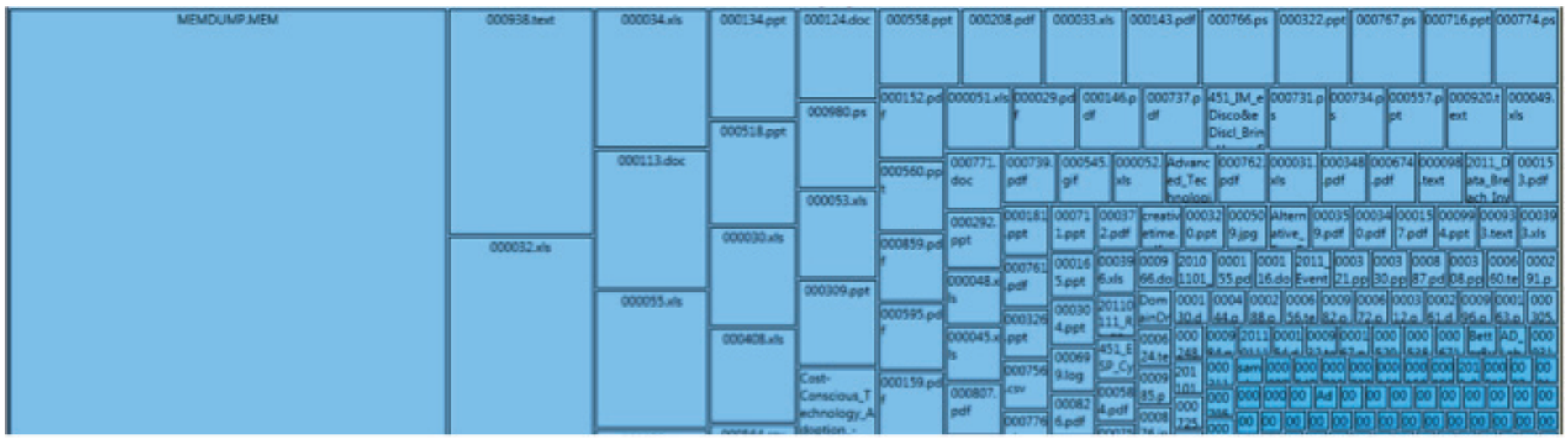
FTK5 (3/4)

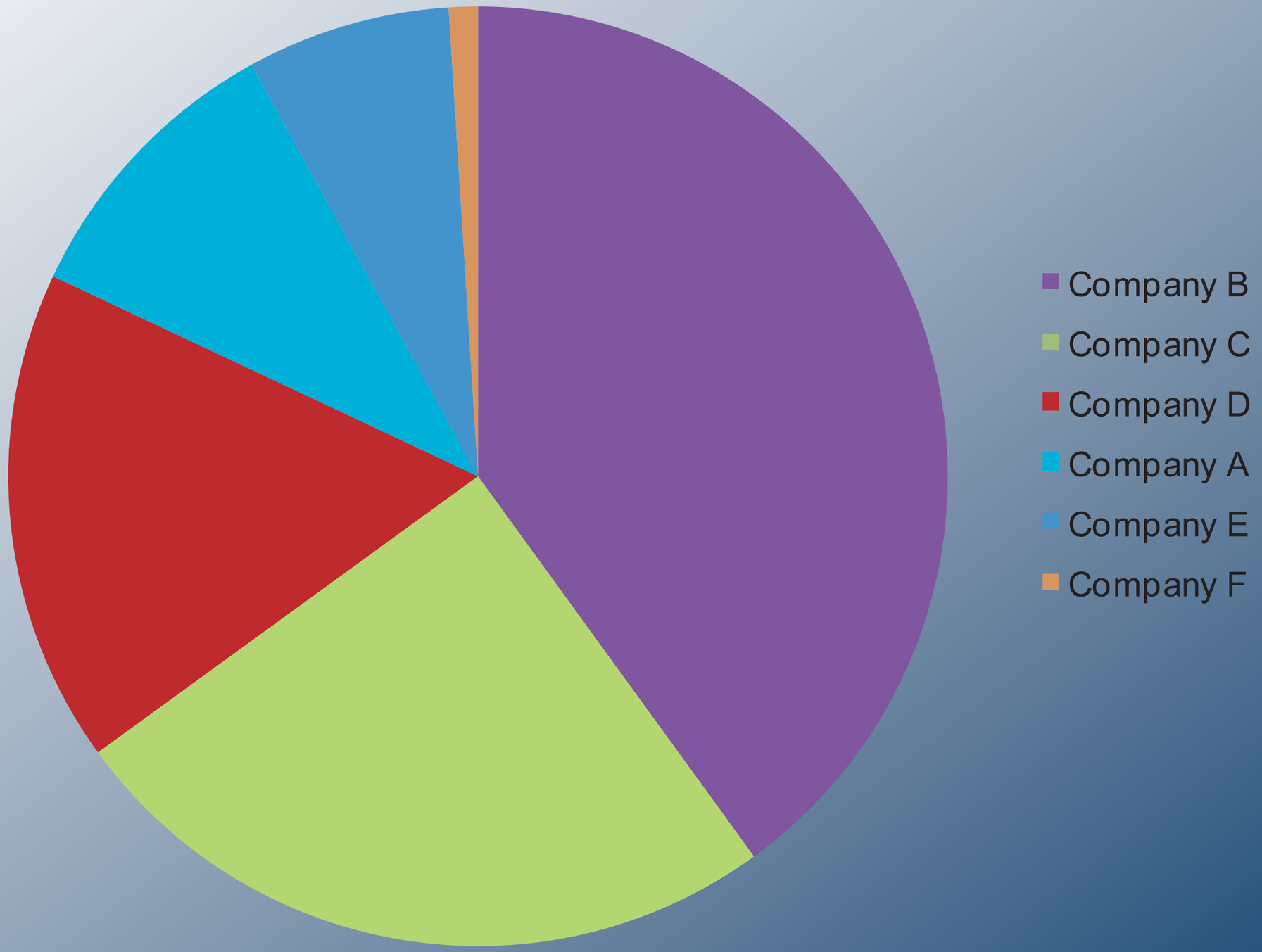




FTK5 (4/4)





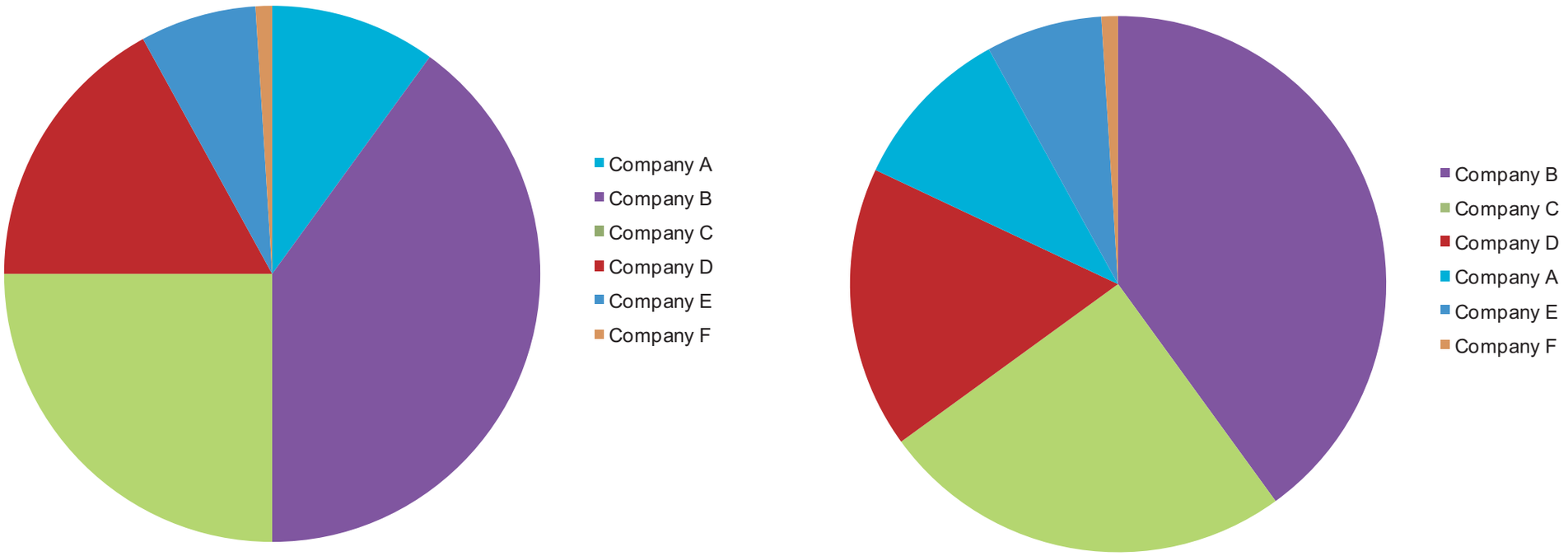


Don’t use pie charts!

Thanks to Stephen Few

Avoid pie charts.

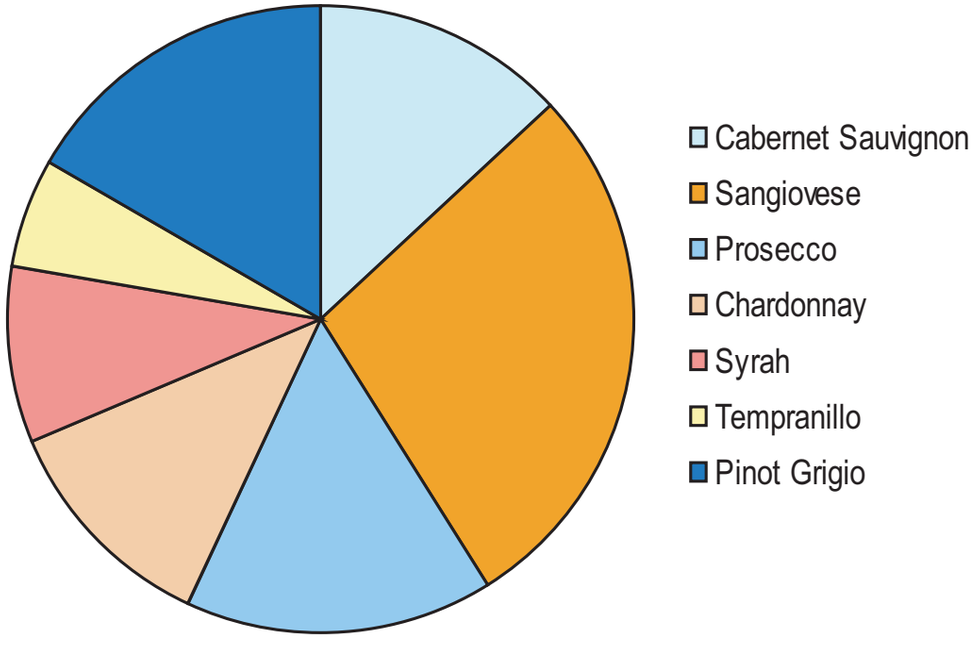
Pie charts are colorful, but are poor for comparing numeric data



Re-ordering a pie chart can influence perception

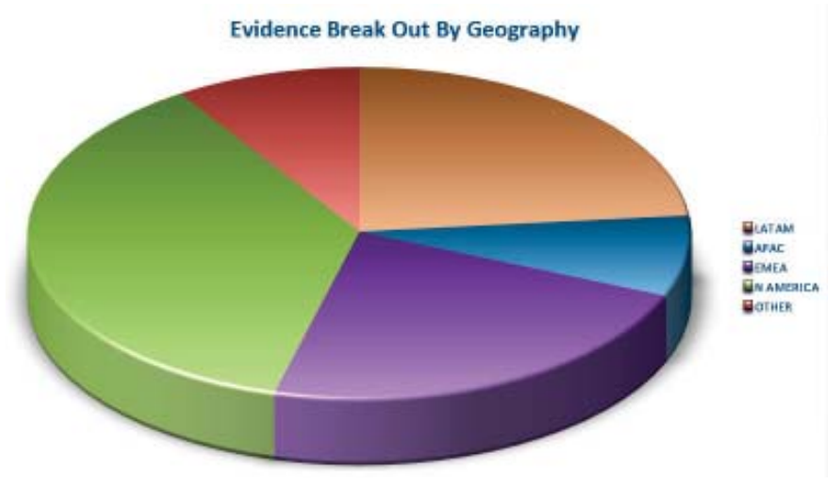
—“Save the Pies for Dessert,” Stephen Few, August 2007

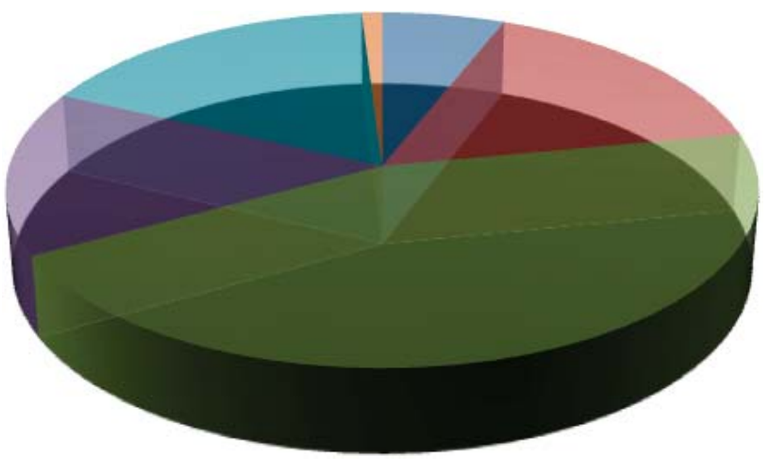
Instead of labeled pie charts, use bar graphs



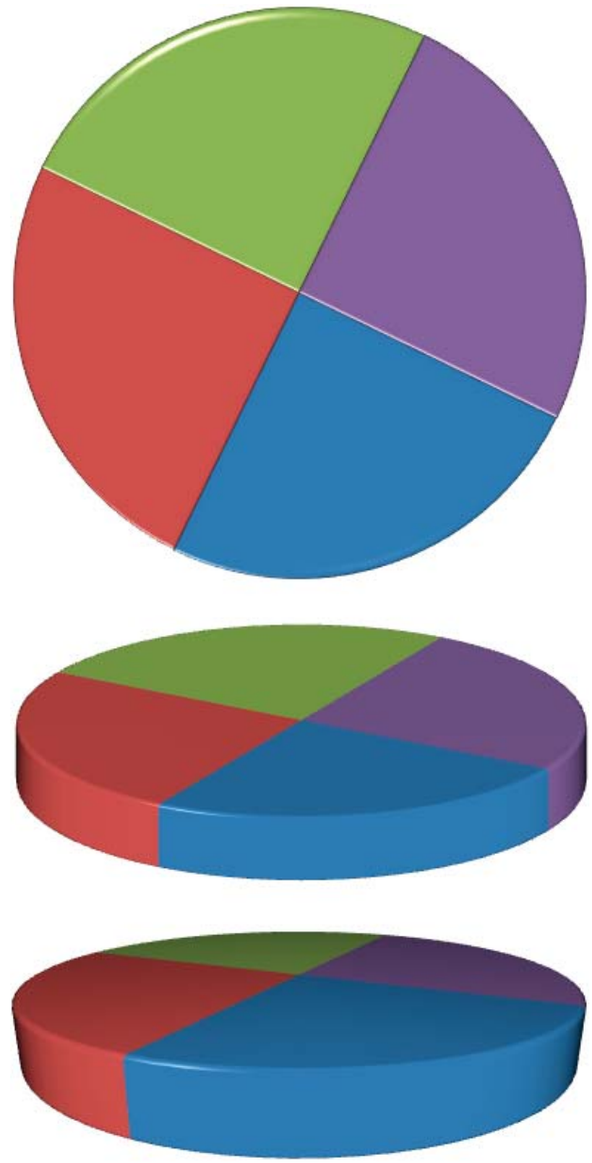
The bar graph makes direct comparison easy!

Never use 3D effects — they distort relationships. (The distortion changes with different 3D projections.)





25:25:25:25



Same data, different projection

There's a lot of work in visualizations — but few translate to open source software.

In most of the academic world, success is a publication.

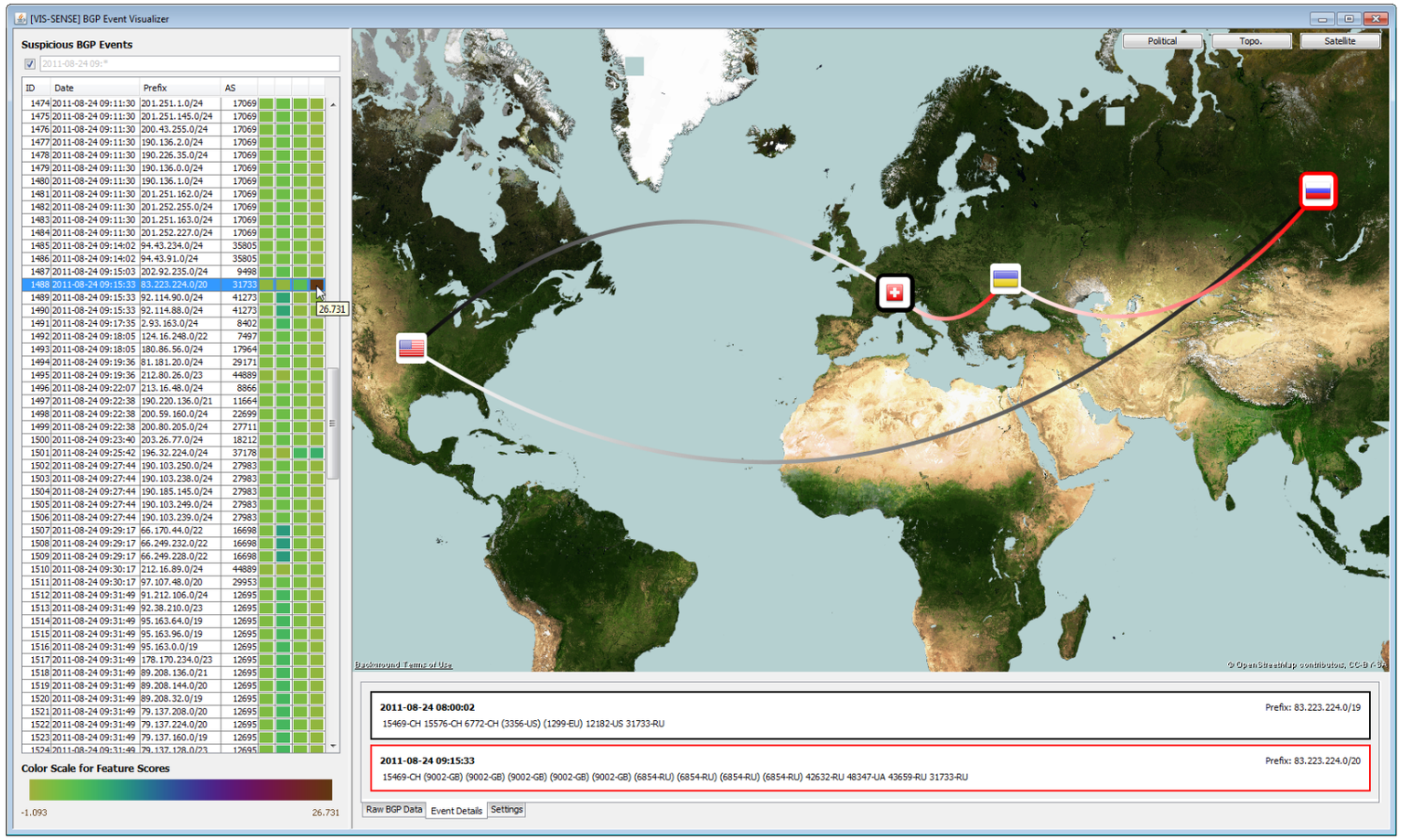


Fig. 5: BGP-Event-Visualization: The pixel visualization on the left acts as an overview to be able to focus on interesting events (*e.g.,* AS31733 with a high Z-Score). The graph visualization with an underlying geographic map reveals details about the selected route. Grey paths are obsolete, red paths are new routings.

"Visual Analytics for BGP Monitoring and Prefix Hijacking Identification"
Fabian Fischer @ University of Konstanz
— http://www.vis.uni-konstanz.de/en/members/fischer/

To sustain forensic visualizations, they must be built into open source software that is used and maintained.